Pós-Graduação em Ciência da Computação

Maria Clara dos Santos Bezerra

MODELOS PARA ANÁLISE DE DISPONIBILIDADE DE ARQUITETURAS DE UM SERVIÇO DE *VOD STREAMING* NA NUVEM

Dissertação de Mestrado

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

RECIFE
2015

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

Maria Clara dos Santos Bezerra

MODELOS PARA ANÁLISE DE DISPONIBILIDADE DE ARQUITETURAS DE UM SERVIÇO DE *VOD STREAMING* NA NUVEM

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
2015

Maria Clara dos Santos Bezerra
Modelos para Análise de Disponibilidade de Arquiteturas de um Serviço de *VoD Streaming* na Nuvem/ Maria Clara dos Santos Bezerra. – RECIFE, 2015-
106 p. : il. (algumas color.) ; 30 cm.

Orientador Paulo Romero Martins Maciel

Dissertação de Mestrado – Universidade Federal de Pernambuco, 2015.

1. Disponibilidade. 2. Computação em nuvem. I. Paulo Romermo Martins Maciel. II. Universidade Federal de Pernambuco. III. Faculdade de xxx. IV. Título

CDU 02:141:005.7

Dissertação de mestrado apresentada por **Maria Clara dos Santos Bezerra** ao programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **Modelos para Análise de Disponibilidade de Arquiteturas de um Serviço de *VoD Streaming* na Nuvem**, orientada pelo **Prof. Paulo Romero Martins Maciel** e aprovada pela banca examinadora formada pelos professores:

Prof. Ricardo Massa Ferreira Lima
Centro de Informática - UFPE

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

Prof. Paulo Romero Martins Maciel
Centro de Informática - UFPE

RECIFE
2015

Dedico esta dissertação à minha família.

Agradecimentos

A jornada até aqui foi árdua, mas tornou-se possível graças ao apoio de algumas pessoas. Agradeço aos meus pais, Roberto e Val, pelo apoio e incentivo para que eu pudesse seguir os meus sonhos. À minha irmã Aça (Maria das Graças) que sempre me acompanhou em todos os momentos da minha vida e me inspira com sua dedicação e esforço. À minha irmã Tay (Natália) pelo apoio, zelo e torcida em cada passo meu. Ao meu sobrinho e afilhado Robertinho (Roberto Neto) que tantas vezes me arrancou sorrisos. Ao meu mais novo sobrinho (Tulio Filho), que trouxe ainda mais luz e amor para nossa família. A Keyla e Andrê por me ouvirem sempre que precisei e até mesmo quando eu achava que não precisava. A todos os meus amigos que entendiam os meus tantos momentos de ausência.

Aos meus amigos e companheiros de pesquisas, boas conversas e cafés: Rosangela e Jamilson, cuja participação nesse trabalho foi fundamental. A Airton pelas dicas, conversas e *coffee breaks*. Agradeço também a Verônica, Igor, Eliomar, Julio e Carlos pelas conversas, dicas e parcerias em tantos momentos que passamos. A todos do grupo MoDCS, sempre disponíveis e colaborativos.

Ao meu orientador Paulo Maciel pela paciência, disponibilidade e pelos ensinamentos prestados. À FACEPE pelo suporte financeiro durante esse trajeto.

Um dia me disseram que as nuvens não eram de algodão.

—HUMBERTO GESSINGER (Engenheiros do Hawaii)

Resumo

A computação em nuvem é um paradigma computacional que vem sendo amplamente utilizado ao longo dos últimos anos devido as suas características de provisionamento de recursos de forma escalável, onde o usuário paga apenas por aquilo que consumir. Esse modelo computacional possibilita que diversos serviços sejam ofertados a partir da sua Infraestrutura como Serviço (IaaS - *Infrastructure as a Service*). Porém, a falha de componentes dos recursos da nuvem é algo bastante comum e que afeta diretamente a disponibilidade dos serviços que os utilizam. Garantir alta disponibilidade em serviços na nuvem é um desafio para provedores, que podem utilizar de técnicas como análise de sensibilidade para identificar gargalos de disponibilidade ou ainda fazer uso de mecanismos de tolerância à falhas para atingir melhores resultados de confiabilidade.

Desse modo, esse trabalho tem como proposta realizar a avaliação de disponibilidade em arquiteturas para o provimento do serviço de VoD *streaming* na nuvem, baseado na plataforma *Eucalyptus*. O trabalho está dividido em dois estudos de caso. Primeiro, apresentamos uma arquitetura básica, sem mecanismos de redundância. Através de modelagem hierárquica formada por diagramas de blocos de confiabilidade (RBD - *Reliability Block Diagram*) e cadeias de Markov de tempo contínuo (CTMC - *Continuous Time Markov Chain*), realizamos a avaliação da disponibilidade para essa arquitetura, e em seguida alcançamos a validação do modelo que representa o funcionamento do serviço. Por fim, aplicamos a técnica de análise de sensibilidade paramétrica para identificar gargalos de disponibilidade na arquitetura básica. As análises seguintes que foram esse estudo de caso observam a disponibilidade do serviço em arquiteturas com redundância no modo *warm-stand* guiadas por índices de sensibilidade, apresentando ao final uma comparação dos valores de disponibilidade para cada arquitetura obtida a partir de componentes redundantes. O segundo estudo de caso analisa o comportamento da disponibilidade total do serviço a partir da variação no tempo de ativação dos mecanismos redundantes.

Nossos resultados expressaram que a adoção de componentes redundantes no modo *warm-stand* na infraestrutura da nuvem proporciona ganhos de disponibilidade no serviço analisado. Também observamos que a variação do tempo de ativação do componente de maior confiabilidade tem maior impacto na disponibilidade total do serviço de *VoD streaming*.

Palavras-chave: Computação em Nuvem, Disponibilidade, Modelos Analíticos, Análise de Sensibilidade.

Abstract

Cloud computing is a computational paradigm that has become widely adopted in recent years. This is due to certain characteristics, in particular its ready scalability and the fact that it is generally offered to the client on a pay-per-use basis, through a type of cloud computing called Infrastructure as a Service (IaaS). However, the failure of components within the cloud is a fairly common occurrence that directly impacts on the availability of the service. Cloud providers attempt to meet the challenge of ensuring high availability by employing certain methods, such as sensitivity analysis to identify availability bottlenecks, and the inclusion of fault tolerance mechanisms to achieve greater reliability.

From this perspective the current work proposes an effective availability analysis of a Video-on–Demand (VoD) streaming service based on the Eucalyptus cloud platform. The research was divided into two case studies. In the first study a basic architecture with no redundancy mechanisms was analysed. The methodology involved first modelling the system through a combination of continuous time Markov chains (CTMC) and reliability block diagrams (RBD), then performing an availability analysis, thereby achieving a model validation that represented the system's behavior. A sensitivity analysis technique was also applied to identify availability bottlenecks in this basic architecture. Subsequent studies analysed availability in increasingly redundant architectures through the inclusion of warm-standby mechanisms. Changes to the basic design were guided by the sensitivity indices, and at the conclusion of the work a comparison was made between the availability values obtained for each system. A further case study analysed the service availability by making variations to the mean time of activation of the redundant mechanism.

Results confirmed that the deployment of redundancy in cloud infrastructures in the form of warm standby mechanisms does lead to gains in system availability. It was also established that varying the activation time of the most reliable component in the system had a greater impact on the overall availability of the VoD service.

Keywords: Cloud Computing, Availability, Analytical Models, Sensitivity Analysis.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

Sumário

1 Introdução

Este capítulo realiza uma breve introdução à computação em nuvem, com relação a *software* e infraestrutura como serviço, além de apresentar conceitos relativos a dependabilidade, que serão tratados ao decorrer da dissertação. Em seguida, nós descrevemos o foco dessa pesquisa a partir da apresentação da motivação e justificativa, bem como dos objetivos esperados e contribuições dadas, além da estrutura da dissertação.

O entretenimento através da Internet vem crescendo mundialmente a cada ano a uma taxa média de 5,7%, com previsão de receita de US$ 2,2 trilhões em 2017 (PWC, 2014). Tal crescimento se justifica pela expansão do acesso à Internet, pelo desenvolvimento de redes de dados móveis como o 4G, e pela explosão no uso de smartphones no mundo todo (LIU; LEE, 2014). O entretenimento através do *streaming* de mídias, como músicas e vídeos, é um dos mais comuns, principalmente devido a grandes provedores desse tipo de serviço como *YouTube* (YOUTUBE, 2014a) e *Netflix* (NETFLIX, 2014a). O *YouTube* recebe mensalmente a visita de mais de um bilhão de usuários únicos, registrando a visualização de mais de seis bilhões de vídeos (YOUTUBE, 2014b). Além disso, diariamente milhões de novas inscrições são realizadas no *YouTube*, e esse número aumentou mais de quatro vezes em comparação ao ano de 2013 (YOUTUBE, 2014b). Já o *Netflix* concentra mais de 53 milhões de usuários pagantes distribuídos em cerca de 50 países, que consomem mais de dois bilhões de horas de conteúdo mensalmente (NETFLIX, 2014b). O consumo de vídeo em 2010 era responsável por 40% de todo o tráfego da Internet, passando para 50% em 2012 com a expectativa de alcançar 62% em 2015 (HWANG et al., 2013).

Outro nicho que vem observando de perto os benefícios da disseminação da Internet, é o de ensino à distância. Só em 2013 foram registradas 2391 instituições ofertando ensino superior à distância para 7.305.977 alunos, um aumento de 3,8% em comparação ao ano anterior. Destes números, 87,41% das instituições eram da iniciativa privada, barganhando 73,54% dos alunos matriculados nessa modalidade de ensino (EDUCAçãO, 2015). O ensino/aprendizado à distância de maneira informal também é um mundo promissor, onde aulas, tutoriais, dicas, mini-cursos, paletras e outros tipos de conteúdo são distribuídos de forma gratuita através da Internet. Uma busca simples no *YouTube*, por exemplo, revela que existem mais de 9.710.000

vídeos relacionados ao termo "aula"de forma gratuita (YOUTUBE, 2015). Já outros serviços como o *UStream* (USTREAM, 2015) e *LiveStream* (LIVESTREAM, 2015), possibilitam que o usuário realize o *streaming* de conteúdos diversos, como transmissão de eventos, *workshops*, congressos, palestras, dentre outros.

Um grande facilitador para o provimento desses tipos de serviços é a computação em nuvem, que é um paradigma computacional capaz de prover acesso sob demanda a recursos computacionais compartilhados, tais como infraestrutura de rede, armazenamento, sistema operacional e aplicações (STANDART; TECHNOLOGY'S, 2013). Esse conjunto de recursos é tipicamente utilizado como um modelo de serviço onde o usuário paga apenas por aquilo que consumir (*pay-per-use*), e o provedor encarrega-se de garantir escalabilidade, segurança, disponibilidade e confiança, baseando-se no que foi firmado nos acordos de nível de serviço (CARDOSO, 2010; J. ARAUJO R. MATOS, 2011; CAROLAN et al., 2009a; ARMBRUST et al., 2010a). Uma vez que esses recursos podem ser adquiridos e entregues com o mínimo de esforço de gerenciamento, os administradores passam a focar nos seus modelos de negócio sem que haja maiores preocupações com detalhes de infraestrutura (STANDART; TECHNOLOGY'S, 2013). Além disso, a computação em nuvem pode ser entregue em três tipos principais de serviço: *Software as a Service (SaaS)*, *Plataform as a Service (PaaS)*, e *Infraestructure as a Service (IaaS)*, (STANDART; TECHNOLOGY'S, 2013). Assim, provedores de IaaS como as plataformas *open-source Eucalyptus* (EUCALYPTUS, 2014a), *OpenNebula* (OPENNEBULA, 2014) e *Nimbus* (NIMBUS, 2014) permitem a criação e gerenciamento de máquinas virtuais em nuvens híbridas e privadas. Isso significa que através desses serviços é possível que o usuário possa ter sua própria coleção de recursos (como *hardware*, armazenamento e rede) através de uma interface de autosserviço e de escalabilidade dinâmica de acordo com a necessidade (EUCALYPTUS CLOUD COMPUTING PLATFORM - ADMINISTRATOR GUIDE, 2010).

A maioria dos serviços *online* utiliza da computação em nuvem para garantir disponibilidade e entrega do serviço apropriado para os seus usuários (ZHU et al., 2011). Sistemas que envolvem transmissão de mídias, a exemplo de jogos e *streaming*, necessitam de alta disponibilidade, visto que a indisponibilidade é um ponto que afeta diretamente o usuário final (WAN et al., 2008). No entanto, como em todo sistema, mesmo que esses serviços sejam executados através de recursos computacionais poderosos, a disponibilidade do sistema pode ser afetada por diversos eventos por tempo indeterminado, como falhas de *hardware*, interrupções programadas para manutenção, mudança de equipamentos, *bugs* no sistema e atualizações. Mecanismos de tolerância à falha são abordagens importantes para lidar com esses tipos de limitações de confiabilidade de *hardware* e *software*.

Mecanismos de tolerância à falha destinam-se a preservar a prestação do serviço correto diante da ocorrência de eventos de falha (AVIZIENIS et al., 2001). Tais mecanismos geralmente são implementados por detecção de erros e suas respectivas recuperações. Um mecanismo de tolerância à falha bastante utilizado para aumento de disponibilidade de infraestruturas é o de redundância de componentes. O uso de tal mecanismo consiste na adoção de componentes de

baixo custo, onde geralmente apresentam uma disponibilidade individual baixa, para que possam prover alta disponibilidade quando agregados a outros componentes (BAUER E; ADAMS, 2012). Entretanto, uma vez que redundância consiste na adoção de equipamentos, é natural que a infraestrutura que a utilize sofra um aumento nos custos de sua operação (FIGUEIREDO et al., 2011).

Essa dissertação propõe uma análise de disponibilidade de um serviço de vídeo *streaming* na nuvem, baseado na plataforma *Eucalyptus*, considerando três arquiteturas diferenciadas pelo uso da adoção do mecanismo de replicação *warm-stand*. Para representar cada arquitetura proposta, é utilizada a abordagem de modelagem através de diagramas de bloco de confiabilidade *(Reliability Block Diagrams (RBD))* e cadeias de Markov de tempo contínuo *(Continous Time Markov Chain (CTMC))*, necessária também para o alcance e comparação dos resultados de disponibilidade obtidos em cada cenário. Através dos modelos analíticos alcançados, é possível obter as expressões de disponibilidade para cálculo da disponibilidade. Além disso, a técnica de análise de sensibilidade paramétrica é utilizada para identificar gargalos de disponibilidade em cada arquitetura. A fim de reforçar a análise das arquiteturas, é feito um levantamento do custo médio para cada cenário.

1.1 Motivação e Justificativa

O provimento de serviços na nuvem demanda confiabilidade, alta disponibilidade e segurança da informação, e mesmo com a evolução das tecnologias envolvidas, é difícil garantir que um serviço não esteja suscetível à falhas. Interrupções nesses tipos de serviços comprometem o seu funcionamento e podem ocasionar insatisfação dos clientes e prejuízos financeiros expressivos, principalmente com serviços que exigem um tempo de resposta rápido, como é o caso dos jogos e do *streaming* de mídias. Soluções a partir de *software* tem sido adotadas com o intuito de utilizar a virtualização para fornecer serviços de *IaaS* na nuvem, visando alcançar confiabilidade dos dados, alta disponibilidade do serviço, segurança, otimização do uso de recursos, redução de custos, dentre outros benefícios (KIM; MACHIDA; TRIVEDI, 2009). A plataforma *Eucalyptus* é um exemplo desses tipos de soluções, e foi desenvolvida com o intuito de construir nuvens privadas e híbridas no modelo de infraestrutura como serviço (*IaaS*), controlando grandes coleções de recursos como rede e armazenamento (EUCALYPTUS, 2014b). Mesmo com o suporte de *softwares* como o *Eucalyptus*, falhas ou atualizações de *software*, manutenções planejadas e troca de equipamentos podem afetar a confiabilidade e/ou a disponibilidade do serviço.

Um serviço de vídeo *streaming* na nuvem que apresente um alto ou constante índice de falha, confiabilidade e segurança, poderá oferecer uma experiência negativa ao usuário, acarretando em prejuízos ao provedor do serviço. O usuário conta com diversas maneiras de criticar um serviço ruim na Internet, e inclusive tem a possibilidade de utilizar serviços concorrentes em questão de poucos segundos. Assim, um sistema de entretenimento na nuvem,

como é o caso dos serviços de vídeo *streaming*, estão vulneráveis a forte rejeição dos usuários caso não ofereçam um serviço de qualidade e de alta disponibilidade, principalmente no caso de entidades educacionais que realizam atividades à distância.

O uso de processos, técnicas, métodos e modelos é muito importante em estudos de avaliação de disponibilidade de sistemas, uma vez que tal avaliação pode promover melhorias na qualidade do serviço provido e no planejamento de infraestruturas. Além disso, a utilização de modelos para representar o funcionamento de sistemas permite resultados confiáveis a um custo muito baixo, visto que não é necessário construir um sistema real para analisá-lo. Da mesma forma, análises de sensibilidade indicam os índices que podem influenciar ganhos em desempenho e disponibilidade em sistemas e infraestruturas, dando *insights* para a elaboração de sistemas e infraestruturas mais disponíveis.

O estudo desenvolvido nessa dissertação almeja fornecer subsídios aos provedores de serviços multimídias, mesmo que de pequeno porte, para que possam planejar e elaborar arquiteturas de nuvem de forma mais eficiente, guiando para o provimento de melhor custo benefício, considerando valores de disponibilidade e de *downtime* anual.

1.2 Objetivos

O principal objetivo deste trabalho é propor um conjunto de modelos para avaliação de disponibilidade de um serviço de vídeo *streaming* sob demanda (*Video on Demand (VoD)*) na nuvem baseado na infraestrutura da plataforma *Eucalyptus*. Essa avaliação combina modelos hierárquicos, análise de sensibilidade e mecanismos de replicação para cada cenário alcançado, com o objetivo de analisar os resultados de disponibilidade para cada proposta. Para que o objetivo geral possa ser alcançado, é necessário cumprir os seguintes objetivos específicos:

- Montar um serviço de vídeo *streaming* em uma infraestrutura privada de nuvem;
- Elaborar modelos para avaliação de disponibilidade de arquiteturas para o provimento de serviço de vídeo *streaming* na nuvem, baseadas na plataforma *Eucalyptus*;
- Validar o modelo através de medições no sistema real baseado em estados que representa o funcionamento do serviço;
- Realizar análises de sensibilidade paramétrica na arquitetura básica alcançada;
- Com base nos resultados de análise de sensibilidade, propor novas arquiteturas com o objetivo de identificar melhorias na disponibilidade do serviço;
- Aplicar análise de sensibilidade nas variações das arquitetuas alcançadas.

1.3 Trabalhos Relacionados

Nos últimos anos, vários trabalhos foram desenvolvidos com o intuito de avaliar as vantagens da adoção de mecanismos de tolerância a falhas e o uso de modelagem hierárquica, a fim de comparar resultados como disponibilidade em diferentes cenários. Além disso, muitos trabalhos tem utilizado dessa abordagem para representar arquiteturas da computação em nuvem, permitindo comparações entre soluções considerando, por exemplo, resultados de disponibilidade a partir da adoção de componentes redundantes. Os trabalhos descritos a seguir apresentam abordagens relacionadas ao desta pesquisa.

Em (DANTAS et al., 2012), os autores investigaram os benefícios do mecanismo de replicação *warm-stand* em um ambiente de computação em nuvem. Para isso, os autores utilizaram duas arquiteturas de nuvens privadas, sendo uma básica e outra redundante e, através de cadeia de Markov e dos parâmetros de falha e reparo dos componentes de cada arquitetura, alcançaram as métricas de dependabilidade de cada cenário. Com isso, foi possível identificar que a arquitetura redundante apresentou os melhores resultados de disponibilidade, graças a replicação do componente mais crítico do cenário. Os modelos apresentados neste trabalho citado serviram de base para o desenvolvimento desta dissertação. Enquanto que os autores trataram de uma replicação de componentes via *warm-standby* considerando uma infraestrutura de nuvem privada, este trabalho considera tanto a infraestrutura quanto um serviço de *VoD streaming*. Além disso, o artigo citado não utilizou nenhuma técnica de análise de sensibilidade para compor suas arquiteturas redundantes, diferente desta dissertação, que utliza da técnica de análise de sensibilidade paramétrica para guiar as variações das arquiteturas propostas.

Em uma pesquisa semelhante, (MATOS et al., 2012a) investigou a disponibilidade de redes de dados a partir da inclusão de um mecanismo de redundância em diversos cenários através da solução numérica-analítica de cadeia de Markov. Além disso, o autor aplicou análise de sensibilidade diferencial para investigar o impacto dos parâmetros dos componentes na disponibilidade total do sistema. Os trabalhos de (DANTAS et al., 2012; MATOS et al., 2012a) citados utilizaram da abordagem de modelagem hierárquica para representar suas arquiteturas e para alcançar os resultados de disponibilidade de cada uma delas. Em (CHUOB; POKHAREL; PARK, 2011) os autores realizaram um experimento para uma plataforma de Governo Eletrônico em uma infraestrutura de computação em nuvem baseada na plataforma *Eucalyptus*, observando o comportamento do ambiente visando alcançar parâmetros de falha e reparo com o intuito de obter resultados de disponibilidade através de modelagem hierárquica. A partir das observações, os autores consideraram que os componentes *Cluster Controller* e *Node Controller* são os mais críticos da infraestrutura da nuvem, portanto, a partir desses dois componentes, é possível obter melhores resultados de disponibilidade. A diferenta destes trabalhos para o trabalho proposto, é que esta pesquisa combina o uso de modelos hierárquicos e analisa um serviço no ambiente de nuvem *Eucalyptus*, obervando a disponibilidade a partir da adição de componentes redundantes.

(MATOS JUNIOR et al., 2011) investigou a disponibilidade de redes de computadores

com componentes redundantes. Para identificar gargalos de disponibilidade do sistema, foi utilizado análise de sensibilidade. Cadeia de Markov foi utilizado para avaliação analítica de cenários complexos. (MATOS et al., 2012b) propõe um método baseado em análise de sensibilidade paramétrica através de modelos Markov de Markov, para avaliar os parâmetros que merecem mais atenção para que a disponibilidade do sistema seja melhorada. Estes trabalhos utilizaram, além de modelos hierárquicos, técnicas de análise de sensibilidade para identificar gargalos de disponibilidade.

Em (GHOSH et al., 2014), o autor realizou análises de disponibilidade em uma infraestrutura na nuvem, onde máquinas físicas estão agrupadas em três tipos de conjuntos de provisionamento de recursos: *hot* (em execução), *warm* (ligado, mas não operante) e *cold* (desligado), divisão realizada com base no consumo de energia e nas características de aguardo no provisionamento de recursos. Então, é feita uma abordagem por modelagem estocástica para análise de dependabilidade desses sistemas na nuvem de provisionamento de IaaS nos três modos acima citados, através de cadeia de Markov.

Em (KIM; MACHIDA; TRIVEDI, 2009) os autores apresentaram um modelo de disponibilidade para um sistema virtualizado e um sistema não-virtualizado, através de modelagem analítica hierárquica por árvores de falhas e CTMC, considerando taxas de falha e reparo de *hardware* e *software*. Com os modelos, os resultados de disponibilidade, inatividade em minutos por ano e disponibilidade orientada à capacidade foram obtidos. Os autores ainda utilizaram a técnica de análise de sensibilidade. Em (FIGUEIREDO et al., 2011) os autores investigaram seis diferentes arquiteturas de infraestrutura para *data centers*, e utilizaram modelagem RBD para representar tais arquiteturas e para alcançar índices de confiabilidade (RI). As arquiteturas são compostas através da adoção de componentes redundantes, com o objetivo de obter melhorias na disponibilidade e na confiabilidade de cada cenário. Ao final, os autores combinam os índices de confiabilidade alcançados com o Custo Total de Aquisição de cada infraestrutura avaliada.

(MELO et al., 2013) apresentou um modelo de disponibilidade baseado em RBD e SPN para avaliar a utilização do mecanismo de *live migration* em um ambiente de computação em nuvem com rejuvenescimento baseado em tempo. O principal objetivo desse trabalho era avaliar o impacto da disponibilidade em diferentes políticas de rejuvenescimento. Em (KHAZAEI et al., 2012) submodelos foram elaborados para representar diferentes passos de serviços em um centro de nuvem complexo e, ao topo dos sub-modelos, modelos estocásticos de disponibilidade e de desempenho foram utilizados para obter os efeitos de falha e reparo dos recursos.

A partir do levantamento dos trabalhos relacionados, a Tabela 1.1 a seguir realiza uma relação entre o que foi levantado dentre eles, e o trabalho aqui proposto. O objetivo é comparar este trabalho com os demais, em termos de limitações e características. É possível notar que todos os trabalhos fizeram uso de modelos analíticos como RBD, CTMC e/ou SPN. Alguns utilizaram apenas uma técnica, mas outros combinaram diferentes técnicas de modelagem, assim como o nosso trabalho proposto. Com relação ao contexto, observamos que prevaleceu o de intraestruturas de nuvem, mas não considerando o uso de algum tipo de serviço nessas infraes-

truturas - caso proposto nesta dissertação. As métricas de disponibilidade e indisponibilidade anual foram utilizadas em termos de análise de propostas, bem como o que realizamos no decorrer deste trabalho. Observa-se que dos trabalhos listados, apenas três consideraram algum tipo de replicação em suas propostas, técnica que foi adotada neste trabalho. Com relação à análise de sensibilidade, destacamos o uso das variações dos parâmetros através da técnica de análise paramétrica - que foi escolhida para guiar as propostas deste estudo. É observado que este trabalho proposto contempla o uso de várias técnicas, que foram observadas de forma distribuída ao decorrer dos trabalhos relacionados. Com relação às limitações deste trabalho, vale ressaltar que alguns pontos desta dissertação (como custo, desempenho e planejamento) não foram trabalhados durante o desenvolvimento da pesquisa, visto que isso poderia causar um distanciamento entre tais pontos e o propósito deste trabalho (disponibilidade, componentes redundantes e análise de sensibilidade).

Desta forma, a importância deste trabalho é observada quando em comparação aos trabalhos relacionados, justamente por poder reunir esse conjunto de técnicas de análise. Durante o decorrer do nosso trabalho, consideramos modelos analíticos (hierárquico e por estados) analisando um serviço de *VoD streamind* numa plaforma de nuvem privada, analisando resultados de disponibilidade a partir de componentes redundantes propostos a partir de análise de sensibilidade paramétrica, além do processo de validação do modelo.

Tabela 1.1: Trabalhos Relacionados

Referências	Modelos Analíticos	Contexto	Análise de Dependabilidade	Tipo de Redundância	Análise de Sensibilidade
(DANTAS et al., 2012)	RBD, CTMC	Eucalyptus	Disponibilidade e Indisponibilidade	*Warm*	Não
(MATOS et al., 2012a)	CTMC	VMs no Eucalyptus	Disponibilidade	Não	Paramétrica e Diferencial
(CHUOB; POKHAREL; PARK, 2011)	CTMC	E-government data center	Disponibilidade e Indisponibilidade	Não	Não
(MATOS JUNIOR et al., 2011)	CTMC	Redes de Computadores	Disponibilidade	*Warm*	Paramétrica
(GHOSH et al., 2014)	CTMC, SPN	Nuvem de IaaS	Disponibilidade e Indisponibilidade	*Hot, warm, cold*	Paramétrica
(KIM; MACHIDA; TRIVEDI, 2009)	CTMC	Sistema virtualizado e não-virtualizado	Disponibilidade e Indisponibilidade	Não	Paramétrica
(FIGUEIREDO et al., 2011)	RBD	Infraestrutura de Data Center	Disponibilidade	Não	Não
(MELO et al., 2013)	CTMC, RBD, SPN	Eucalyptus	Disponibilidade e Indisponibilidade	Não	Paramétrica
(KHAZAEI et al., 2012)	CTMC	Data Center	Disponibilidade	Não	Não

1.4 Estrutura da Dissertação

O Capítulo 2 aborda a fundamentação teórica do trabalho, apresentando temas como a computação em nuvem, sua estrutura e alguns tipos de serviços possíveis através dessa tecnologia. É feita uma abordagem a respeito de dependabilidade e, mais especificamente, sobre disponibilidade, que é o foco esse trabalho. Também são apresentadas técnicas para avaliação de dependabilidade através de modelagem por meio de cadeia de Markov e diagrama de blocos de confiabilidade. Após isso, o conceito de análise de sensibilidade é abordado apresentando algumas técnicas de análise. Adiante, este capítulo ainda aborda a plataforma *Eucalyptus*, mostrando a sua arquitetura, seus componentes e seu funcionamento.

O Capítulo 3, seguinte, refere-se à plataformas de *VoD streaming*, trazendo alguns conceitos a respeito desse tipo de serviço e apresentado o serviço que foi utilizado para realizar essa pesquisa. Na sequência, a metodologia utilizada para guiar essa pesquisa é apresentada no Capítulo 4, descrevendo os passos para a realização da análise de disponibilidade das arquiteturas estudadas. Esse capítulo também trata das variações das arquiteturas alcançadas, assim como os modelos analíticos desenvolvidos para a análise de disponibilidade dessas arquiteturas. Esse capítulo também apresenta a validação dos modelos alcançados.

Tomando por base os modelos propostos, o Capítulo 5 apresenta os estudos de casos para avaliação de disponibilidade das arquiteturas obtidas, verificando o ganho de disponibilidade em cada variação. Por fim, o Capítulo 6 finaliza este trabalho, apresentando as principais contribuições e sugerindo possíveis trabalhos futuros.

2

Fundamentação Teórica

Este capítulo concentra os conceitos básicos a respeito dos temas estudados nesta dissertação. É válido ressaltar que aqui subsídios são fornecidos para o entendimento das técnicas aplicadas no decorrer do trabalho, e que cada tema pode ser melhor explorado a partir das referências mencionadas. Este capítulo está estruturado da seguinte maneira: primeiro, os conceitos básicos de dependabilidade são apresentados e, em seguida, algumas técnicas de modelagem de dependabilidade são explicadas, assim como a técnica de injeção de falhas para validação de modelos. Por fim, as características e os fundamentos relacionados à computação em nuvem são apresentados, bem como os da plataforma *Eucalyptus*.

2.1 Avaliação de Desempenho de Sistemas

Há várias definições de dependabilidade. Uma das definições diz que dependabilidade do sistema pode ser entendido como a capacidade de oferecer uma funcionalidade específica, que pode ser justificadamente confiável (AVIZIENIS et al., 2004), ou ainda, que o sistema irá executar ações especificadas ou apresentar resultados específicos de maneira confiável e em tempo oportuno (PARHAMI, 1988). Dependabilidade abrange medidas como a disponibilidade, confiabilidade e segurança.

Devido à prestação de serviços onipresente na Internet, a dependabilidade tornou-se um atributo de interesse principal em *hardware*/*software* de desenvolvimento, implantação e operação (MACIEL et al., 2011). Como a computação em nuvem é um paradigma de computação distribuída em grande escala e suas aplicações são acessíveis em qualquer lugar e em qualquer momento, a alta dependabilidade de sistemas na nuvem está se tornando mais importante e mais difícil de se alcançar (SUN et al., 2010).

(AVIZIENIS et al., 2004) realiza uma abordagem sistemática dos conceitos de dependabilidade, dividindo-o em três partes: as ameaças, os atributos e os meios pelos quais a dependabilidade é alcançada, como mostra a Figura 2.1:

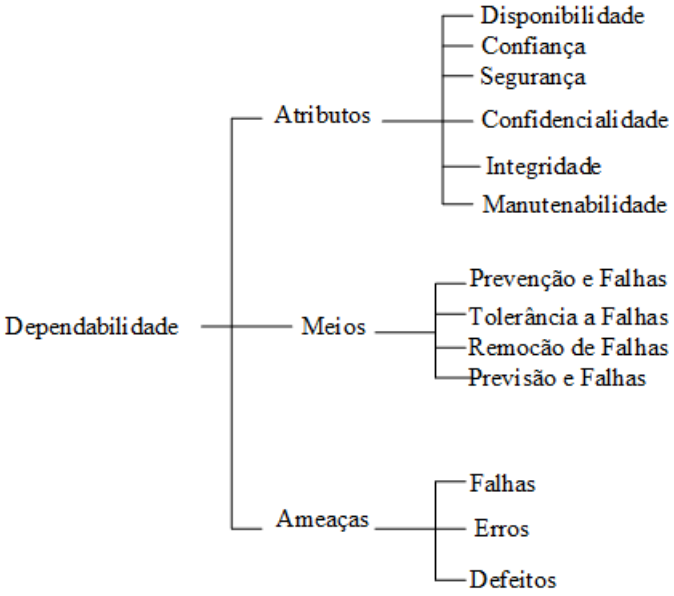


Figura 2.1: Árvore de Dependabilidade.

Os **atributos** possibilitam que medidas quantitativas, que são cruciais, sejam obtidas para a análise dos serviços ofertados. Os **meios** são os caminhos pelos quais a dependabilidade é alcançada. As **ameaças** referem-se às falhas, aos erros e aos defeitos.

Uma falha ocorre quando um erro atinge o serviço alterando-o segundo suas características (LAPRIE, 1992). Falha nos componentes (*hardware e/ou software* que formam o serviço na nuvem podem afetar a disponibilidade do sistema. Assim, a utilização de técnicas como replicação de componentes críticos em determinadas arquiteturas de muitos sistemas na nuvem pode garantir os aspectos de dependabilidade apontados e descritos adiante.

Dependabilidade é um conceito integrativo que engloba os seguintes atributos básicos: disponibilidade, confiabilidade, segurança, confidencialidade, integridade e manutenabilidade (AVIZIENIS et al., 2001). Os detalhes de cada atributo são dados a seguir.

Confiabilidade

O atributo confiabilidade é definido como sendo a probabilidade de que um sistema irá executar a função pretendida durante um período de tempo de funcionamento sem qualquer falha (MUSA, 2004). Dessa maneira, a confiabilidade do sistema pode ser interpretada como uma medida para a continuidade do serviço e, para que a confiabilidade de um sistema seja descrita, é necessário tomar conhecimento da sua configuração, do estado em que o define como operacional e das regras de operações (KUO; ZUO, 2003).

Matematicamente, a função de confiabilidade R_t é a probabilidade de que o sistema irá funcionar corretamente, sem a ocorrência de falhas, no intervalo de tempo de 0 até t (KUO; ZUO, 2003; MACIEL et al., 2011), como expressa a Equação 2.1:

$$R(t) = P(T > t), t \geq 0 \tag{2.1}$$

Onde T indica uma variável aleatória, que representa o tempo de falha ou o tempo para falha.

Disponibilidade

A importância de se ter um sistema disponível é observada por grande parte de usuários de sistemas em rede, tais como compras *on-line* e acesso à sua conta bancária as quais têm que estar disponíveis sempre que o usuário necessitar das mesmas. Devido a essa necessidade de obtenção de recursos na hora exigida, usuários demandam sistemas que venham atendê-lo a qualquer hora.

Podemos dizer que a disponibilidade de um sistema pode ser entendida como a probabilidade de que ele esteja operacional durante um determinado período de tempo, ou tenha sido restaurado após a ocorrência de um evento de falha. *Uptime* representa o período de tempo em que o sistema está operacional; *downtime* é o período de tempo em que o sistema estará indisponível devido a ocorrência de um evento de falha ou atividade de manutenção preventiva ou corretiva; e o período de tempo de observação do sistema é expresso através da soma entre e *uptime* e *downtime*.

A Equação 2.2 representa a disponibilidade de um sistema expressado através da relação entre Tempo Médio de Falha (*Mean Time To Failure (MTTF)*), e Tempo Médio de Reparo (*Mean Time To Repair (MTTR)*) (MACIEL et al., 2011). A disponibilidade calculada desta forma será um número entre zero e um. Esse valor também pode ser expresso em termos de números de noves. Por exemplo, se a disponibilidade do sistema é igual a 0,999876, isso significa dizer que o sistema se encontra funcionando durante 99,9876% do tempo inativo em 0,0124% do tempo observado. O número de noves da disponibilidade pode ser calculado conforme a Equação 2.3. 100 representa o nível de disponibilidade máxima que o sistema pode atingir e A representa a disponibilidade real do sistema.

$$A = \frac{uptime}{uptime + downtime} \tag{2.2}$$

$$N = 2 - log(100 - A) \tag{2.3}$$

Desta maneira, o valor da disponibilidade é obtido através do cálculo dos índices de MTTF e MTTR, onde:

• MTTF: tempo médio para que ocorram falhas no sistema, representado pela Equação 2.4:

$$MTTF = \int_0^\infty R(t)dt \tag{2.4}$$

• MTTR: tempo médio em que o sistema está indisponível por causa da atividade de reparo. Esse valor varia de cada organização, visto que ele está relacionado às políticas de manutenção e características referentes a cada ação específica de reparo. A Equação 2.5 representa o MTTR:

$$MTTR = MTTF \times \frac{UA}{A} \tag{2.5}$$

onde *UA* representa a indisponibilidade do sistema (*Unavailability*), como mostra a Equação 2.6; e A indica a disponibilidade do sistema (*Availability*), como expressa a Equação 2.2 acima.

$$UA = 1 - A \tag{2.6}$$

Desta maneira, tempo de *downtime* anual (*downtime*) do sistema pode ser calculado através da Equação 2.7:

$$D = UA \times 8760(horas) \tag{2.7}$$

A variação da taxa de falha dos componentes de hardware é mostrada na Figura 2.3. Conhecida como curva da banheira, a figura demonstra a taxa de falhas de componentes de hardware em três fases distintas (EBELING, 2004), representadas pelos números 1, 2 e 3.

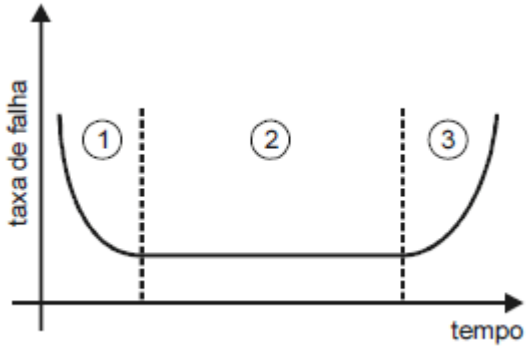


Figura 2.2: Curva da Banheira.

Durante a primeira fase, representada pelo número 1, ocorre um curto período em que a taxa de falha é bastante alta. Falhas ocorridas nesse período são decorrentes de defeitos de fabricação do equipamento. Com o intuito de encurtar esse período, fabricantes submetem os equipamentos a um processo chamado *burn-in*, onde eles são expostos a elevadas temperaturas de funcionamento.

Na segunda fase, indicada pelo número 2, as falhas ocorrem aleatoriamente. Valores de confiabilidade de equipamentos fornecidos por fabricantes aplicam-se a esse período. Durante a fase final, a taxa de falhas cresce exponencialmente. O período de vida útil do equipamento normalmente não é uma constante. Ele depende do nível de estresse em que o equipamento é submetido durante esse período. A disponibilidade do sistema pode ser dividida em três classes (RESNICK, 1996): disponibilidade básica, alta disponibilidade e disponibilidade contínua.

Um sistema de disponibilidade básica não requer quaisquer mecanismos para prevenção ou recuperação rápida de falhas, ou qualquer tipo de redundância. Esses sistemas atendem às expectativas básicas dos usuários, porém se ocorrer um evento de falha ou algum tipo de

manutenção planejada, o sistema ficará indisponível por tempo indeterminado.

A Figura 2.3 representa o cenário de um provedor de sistema em nuvem. Quando há a sobrecarga ou uma falha em um de seus componentes, esse servidor para o serviço, ficando indisponível por algum tempo, retomando as atividades normais depois de realizado uma atividade de reparo.

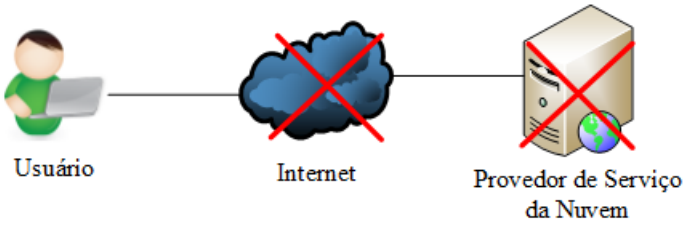


Figura 2.3: Exemplo de Disponibilidade Básica em um Serviço.

Esse tipo mais básico de disponibilidade pode chegar, a três noves de disponibilidade. Assim, em um ano de operação, a máquina pode ficar indisponível por um período de nove horas a quatro dias, somados todos os possíveis períodos de indisponibilidade. No entanto, em sistemas que se deseja ter uma alta disponibilidade, só é possível por intermédio de dispositivos de *software* ou *hardware* que sejam capazes de detectar e recuperar *hosts* que obtiveram falhas, recuperando-os em um menor tempo possível.

A Figura 2.4 demonstra um cenário composto por dois *hosts*, um *switch* e dois roteadores compartilhando *links* redundantes na tentativa de prover os serviços desejados por maior tempo possível (GUIMARÃES et al., 2011). Quando o primeiro *link* falha (*link* 1) o *link* em espera (*link* 2) entra em atividade. Depois da restauração do *link* 1, o sistema volta para seu estado primário. Com esse tipo de serviço o administrador da rede tenta prover o mais próximo de 100% da entrega do serviço

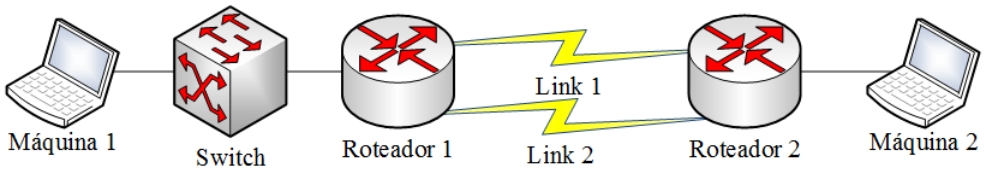


Figura 2.4: Exemplo de Disponibilidade Alta.

A disponibilidade contínua (ou *continuous availability*) visa manter um sistema totalmente disponível. Esse tipo de disponibilidade é muito mais ambicioso quando comparada aos outros tipos de disponibilidade, pois visa à entrega do serviço sem interrupções, o que implica em 24 horas por dia e sete dias por semana durante os 365 dias no ano.

Para prover continuidade do serviço em transferência de dados, podem ser criados centros de dados em pontos estratégicos, espalhados geograficamente. No exemplo de disponibilidade contínua mostrado na Figura 2.5, quando o centro de dados 1 falhar, seja essa falha por problemas de hardware e/ou software de componentes individuais ou ainda pela ocorrência de falha do

centro de dados, o centro de dados 2 poderá assumir os serviços fazendo com que os dados trafeguem ininterruptamente. Assim, manutenções planejadas ou qualquer tipo de operação que cause a parada do sistema, são mascaradas, não interferindo na disponibilidade do sistema, mesmo em caso de falhas catastróficas.

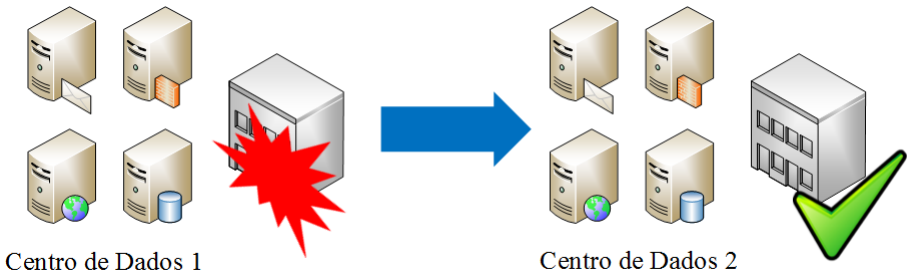


Figura 2.5: Exemplo de Disponibilidade Contínua.

Confidencialidade

Refere-se à ausência de divulgação não autorizada de informação (AVIZIENIS et al., 2001), tendo essas informações que permanecer secretas, sendo acessadas apenas por pessoas autorizadas. As consequências de acessos não autorizados à informações são sérias, afetando não apenas aplicações de segurança nacional, mas afetando também o comércio e a indústria (AVIZIENIS et al., 2001).

Integridade

O atributo integridade é pré-requisito para a confiabilidade, disponibilidade e segurança (AVIZIENIS et al., 2001). A integridade consiste na tentativa de garantir que os dados do sistema não sejam alterados durante uma comunicação. Mecanismos de proteção de integridade podem ser agrupados em duas categorias, que são (VERA-SSIMO; RODRIGUES, 2001):

- Mecanismos de prevenção: como controles de acesso que impeçam a modificação não autorizada de informações e;
- Mecanismos de detecção: responsáveis por detectar modificações não autorizadas enquanto os mecanismos de prevenção estavam falhos.

Manutenabilidade

Normalmente, quando um sistema apresenta uma falha ou não entrega mais os serviços, operações de reparo são utilizadas para que o problema possa ser contornado. Dessa maneira, a manutenabilidade é justamente a capacidade de o sistema sofrer modificações e reparos (TRIVEDI et al., 2009). Esse atributo pode ser descrito como a probabilidade de que um sistema seja reparado após a ocorrência de um evento de falha em um determinado período de tempo.

2.1.1 Meios de Dependabilidade

O desenvolvimento de um sistema de computação confiável requer a utilização combinada de um conjunto de quatro técnicas (AVIZIENIS et al., 2001):

Prevenção a Falhas

Relaciona-se a como prevenir a ocorrência de falhas, e é alcançada por meio de técnicas de controle de qualidade utilizadas durante o projeto e fabricação de hardware e software (AVIZIENIS et al., 2001). Partindo dessa perspectiva, prevenção de falha é a prevenção de ocorrência ou introdução de algum tipo de falha no sistema.

Tolerância a Falhas

Está relacionada a como oferecer um serviço correto mesmo na presença de falhas. É geralmente implementada por detecção de erros e recuperação do sistema subsequente (AVIZIENIS et al., 2001). Essa técnica garante o funcionamento do sistema mesmo na ocorrência de falhas e são baseadas em redundância, adição de componentes no sistema.

Remoção de Falhas

Remoção de falhas é realizada, durante a fase de desenvolvimento e a vida operacional de um sistema, com o objetivo de reduzir o número de falhas e gravidade das falhas no sistema. A remoção de falha durante a fase de desenvolvimento de um sistema consiste em três etapas (AVIZIENIS et al., 2001): verificação, diagnóstico e correção.

A verificação é o processo de verificar se o sistema adere propriedades dadas, ou seja, verificar se as funcionalidades do sistema estão funcionando de forma correta. Se isso não acontecer, passa-se para os passos seguintes: diagnóstico de falha(s) que impediu que as condições de verificação fossem cumpridas, e depois realizar as correções necessárias. Após a correção, o processo de verificação deve ser repetido a fim de verificar que a remoção da falha não teve consequências indesejáveis.

Previsão de Falhas

Estimar o número atual, a incidência futura, e as prováveis consequências das falhas. Assim, estima-se, por avaliação, a presença, a criação e a consequência das falhas. Utilizam-se de técnicas de injeção de falhas e testes de resistência. A previsão de falhas é realizada através da realização de uma avaliação do comportamento do sistema em relação à ocorrência de falha ou de ativação. Essa avaliação tem dois aspectos (AVIZIENIS et al., 2001):

- Qualitativo, ou ordinal, tipo de avaliação que visa identificar, classificar e ordenar os modos de falha, ou as combinações de eventos (falhas de componentes ou condições ambientais) que levam a falha no sistema;

- Quantitativa, ou probabilística, tipo de avaliação que tem como objetivo avaliar, em termos de probabilidades, na medida em que alguns dos atributos de confiabilidade estão satisfeitos.

2.2 Técnicas de Modelagem de Dependabilidade

Existem vários tipos de modelos que podem ser utilizados para a avaliação analítica de dependabilidade. Diagramas de bloco de confiabilidade, árvores de falhas, redes de Petri estocásticas e cadeias de Markov têm sido usados para modelar sistemas tolerantes a falhas e avaliar medidas dependabilidade diferentes. Esses tipos de modelo diferem de um para outro, não só na facilidade de utilização para uma aplicação em particular, mas em termos de poder de modelagem (MALHOTRA; TRIVEDI, 1994).

Eles podem ser classificados em modelos combinatórios e baseados em estado (MACIEL et al., 2011). Modelos baseados em estado podem também se referir como não combinatórios, e modelos combinatoriais podem ser identificados como modelos não baseados em estados. O tipo de modelo baseado em estados é mais utilizado para a representação de sistemas de maior complexidade, representando o comportamento do sistema. Já os modelos combinatoriais, que não são baseados em estados, são utilizados para representar a interação entre os componentes que formam um sistema.

2.2.1 Cadeias de Markov de Tempo Contínuo

Uma Cadeia de Markov é um modelo baseado em estado no qual o estado atual não depende dos estados anteriores para que se conheçam os estados seguintes. Criado para a modelagem de sistemas, onde é possível descrever o funcionamento de um sistema por meio de um conjunto de estados e transições. As cadeias de Markov são modelos estocásticos comumente utilizados para a descrição de análises estatísticas que possuem valores de tempo em seus parâmetros.

Um processo estocástico $X(t), t \in T$ é um conjunto de variáveis aleatórias definidas sobre o mesmo espaço de probabilidades, indexadas pelo parâmetro de tempo ($t \in T$) e assumindo valores no espaço de estados ($_{si} \in S$) (CASSANDRAS; LAFORTUNE, 2008). Assim, se o conjunto T for discreto, ou seja, enumerável, X(t), t = 1, 2, 3..., o processo é dito processo de parâmetro discreto ou tempo discreto. Se T for um conjunto contínuo, tem-se um processo de parâmetro contínuo ou tempo contínuo.

O processo estocástico é classificado como um processo de Markov se, para todo $t_0 < t_1 < ... < t_n < t_{n+1}$ e para todo $X(t_0), X(t_1), X(t_2), ..., X(t_n), X(t_{n+1})$, a distribuição condicional de $X(t_{n+1})$ depender somente do último valor anterior $X(t_n)$ e não dos valores anteriores $X(t_0), X(t_1), ..., X(t_{n+1})$, isto é, para qualquer número real $X_0, X_1, X_2, ..., X_n, X_{n+1}$, $P(X_{n+1} = s_{n+1} | X_n = s_n, X_{n-1} = s_{n-1}, ..., X_0 = s_0) = P(X_{n+1} = s_{n+1} | X_n = s_n)$ (BOLCH et al., 2006).

Uma cadeia de Markov é descrita por uma sequência de varáveis aleatórias discretas, $X(t_n)$, em que t_n pode assumir um valor discreto ou contínuo, isto é, uma cadeia de Markov é um processo de Markov com um espaço de estados discretos.

As Cadeias de Markov representam o comportamento do sistema (falhas e atividades de reparo) pelos seus estados, e a ocorrência de evento é expressada pela transição de um estado para outro (MACIEL et al., 2011). As etiquetas podem ser probabilidades, taxas ou funções de distribuição. A cadeia de Markov constitui um tipo particular de processo estocástico com estados discretos e com o parâmetro de tempo podendo assumir valores contínuos ou discretos (SOUSA, 2009). Portanto, cadeias de Markov de tempo contínuo, as CTMC (*continuous time Markov chains*), possuem transições que podem ocorrer em qualquer instante do tempo, e as de cadeia de Markov de tempo discreto, DTMC (*discrete-time Markov chains*), possuem transições que ocorrem em tempos discretos de tempos.

Sendo assim, a representação de um modelo através do formalismo de cadeia de Markov pode ser interpretada como uma máquina de estados, onde os nós (vértice de um grafo) desta representam os estados, e os arcos representam as transições entre os estados do modelo em cadeia de Markov (MACIEL et al., 2011).

Se o modelo é discreto, a escala de tempo para a transição entre os estados do modelo pode ser de forma contínua (CTMC) ou discreta (DTMC). A transição entre os estados do modelo depende exclusivamente do estado atual deste, sem importar quais formam os estados prévios ou futuros de tal modelo. A taxa (CTMC) ou probabilidade (DTMC) de transição de estados do modelo dá-se obedecendo a uma lei exponencial ou geométrica, respectivamente (MACIEL et al., 2011).

Para representar graficamente um modelo em Cadeia de Markov é feita uma associação entre os estados e em cada transição entre os estados é inserida uma taxa ao modelo de tempo contínuo (CTMC) ou probabilidade para modelos discretos (DTMC). Desta forma, um modelo em cadeia de Markov (CTMC) pode ser representado matematicamente por uma matriz de transição de estados. A probabilidade de cada estado em regime estacionário (solução de um modelo em cadeia de Markov) é a solução do sistema da Equação linear 2.8.

$$Q = \begin{pmatrix} q_{ii} & q_{ij} \\ q_{ji} & q_{jj} \end{pmatrix}, \tag{2.8}$$

Onde Q é a matriz de estados e π (vetor de probabilidade) é o autovetor correspondente ao autovalor unitário da matriz de transição, resultando em um vetor 0. É importante ressaltar que a soma dos elementos do vetor de probabilidade π deve ser igual a 1, ou seja, $\|\pi\| = 1$ (MACIEL et al., 2011). A representação gráfica de uma cadeia de Markov é representada por um diagrama de transições. Assim, podem ser visualizados os estados, sendo representados por círculos, e as transições representadas por arcos, além das taxas e/ou probabilidades das transições.

Para as cadeias de Markov de tempo contínuo, a matriz de taxas é cada elemento não diagonal da linha i e coluna j, onde as mesmas representam a taxa de transição do estado i para o

estado j do modelo. Os elementos diagonais representam o ajuste necessário para que a soma dos elementos de cada linha seja zero. As probabilidades de transição dos estados podem ser calculadas através da Equação 2.9.

$$p_{i,j}(s,t) = PX(t) = j|X(s) = i \tag{2.9}$$

A solução transiente, ou dependente do tempo, é importante quando o sistema a avaliar é dependente do tempo (SOUSA, 2009). Para modelos ergódigos, considerando tempos de execução longos, pode-se mostrar que a probabilidade dos estados converge para valores constantes (HERZOG, 2001). O comportamento transiente da cadeia de Markov nos fornece informações de desempenho e dependabilidade sobre os instantes iniciais do sistema. Assumindo-se que a probabilidade $\pi(t)$ é independente do tempo, isto é, $lim_{t\to\infty}\pi_i(t)$ (homogeneidade), consequentemente, $\pi'(t) = 0$, resultando nas Equações 2.10 e 2.11:

$$\pi Q = 0 \tag{2.10}$$

$$\sum_{i=1}^{N} \pi_i = 1 \tag{2.11}$$

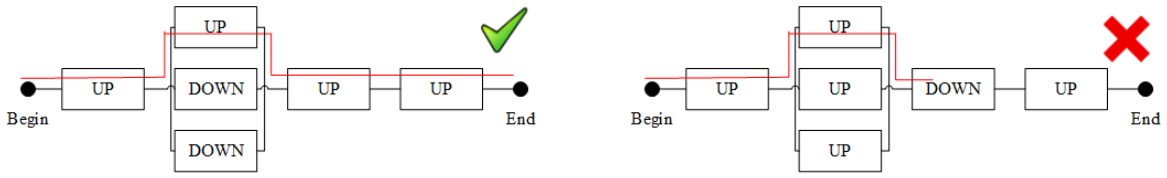
A Equação 2.11 é a condição de normalização, adicionada para assegurar que a solução obtida é um único vetor de probabilidade. A Equação 2.10 tem um conjunto de soluções infinitas. Normalizando as soluções, chega-se a um único vetor de probabilidades.

Desta forma, as cadeias de Markov têm importância fundamental no processo de modelagem de sistemas redundantes e avaliação de dependabilidade nos mais variados sistemas.

2.2.2 Diagramas de Bloco de Confiabilidade

Diagrama de Blocos de Confiabilidade (*Realibility Block Diagram* - RBD) é um tipo de modelo combinatório proposto inicialmente para analisar a confiabilidade de sistemas baseados nas relações de seus componentes (MACIEL et al., 2011). Posteriormente, este formalismo foi estendido para a análise de disponibilidade e manutenibilidade. Nesta Seção, atentaremos para o uso de RBDs para o cálculo de disponibilidade, visto que este é o foco deste trabalho.

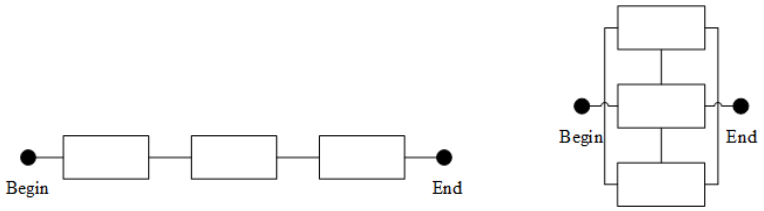
Um RBD é formado pelos seguintes componentes: vértice de origem, vértice de destino, componentes do sistema (representados por blocos), e arcos conectando os blocos e os vértices. Um sistema modelado em RBD está disponível caso haja algum caminho do vértice de origem até o vértice de destino, no qual componentes indisponíveis representam uma interrupção em um segmento do caminho. Na Figura 2.6 ilustramos este fato, onde de um lado mostra que o sistema permanece disponível mesmo com a falha de dois componentes, uma vez que há um caminho contínuo dos vértices de início e fim do RBD. Já na representação do sistema indisponível, a falha do componente é crítica para o sistema, uma vez que ele impede que haja tal caminho no RBD.



(a) Sistema Disponível (b) Sistema Indisponível

Figura 2.6: Exemplo de Disponibilidade em RBD.

Há duas formas principais de arranjar os blocos, como ilustra a Figura 2.7: em série e em paralelo. Em um conjunto de componentes em série, todos os componentes deverão estar operacionais para que conjunto esteja disponível. Em conjunto de componentes em paralelo, por sua vez, é necessário que pelo menos um esteja disponível. Outra estrutura típica é a *k-out-of-n*, que é formada por n componentes, e necessita do funcionamento de pelo menos k componentes para estar operacional (MACIEL et al., 2011).



(a) Componentes em Série (b) Componentes em Paralelo

Figura 2.7: Agrupamentos de componentes em um RBD.

Para avaliar a disponibilidade através de um RBD, introduzimos antes o conceito de função estrutural de um sistema. Considere que um sistema S é formado por n componentes: c_1 , c_2 , c_3 , . . . c_n, e este sistema pode estar em falha ou operacional, de acordo com o estado de seus componentes. O estado de um componente ci é dado pela variável aleatória xi, tal que:

$$x = \begin{cases} 0, & \text{se o componente falhou} \\ 1, & \text{se o componente está operacional} \end{cases}$$

A confiabilidade de dois blocos conectados em série (ver Figura 4.2) pode ser obtida através da Equação 2.12:

$$R_s(t) = R_1(t) \times R_2(t) \tag{2.12}$$

Onde:
$R_1(t)$ é a confiabilidade do bloco 1, e
$R_2(t)$ é a confiabilidade do bloco 2.

De uma forma geral, considerando um sistema com n elementos, a confiabilidade do modelo em série pode ser obtida através da seguinte Equação 2.13:

$$R_s(t) = \prod_{i=1}^{n} R_i(t) \tag{2.13}$$

Onde:
$R_i(t)$ é a confiabilidade do bloco b_i.

A confiabilidade de dois blocos conectados em paralelo pode ser obtida através da seguinte Equação 2.14:

$$R_p(t) = 1 - \prod_{i=1}^{2} (1 - R_i(t)) \tag{2.14}$$

Para esse tipo de sistemas temos que pelo menos um componente deve ser operacional para que todo sistema esteja funcionando. De uma forma geral, considerando n componentes, a confiabilidade do sistema pode ser obtida através da seguinte Equação 2.15:

$$R_p(t) = 1 - \prod_{i=1}^{n} (1 - R_i(t)) \tag{2.15}$$

Onde: $R_i(t)$ é a confiabilidade do bloco b_i.

Modelos RBDs são utilizados em sistemas que contêm módulos independentes, onde cada um pode ser facilmente representado por um bloco de confiabilidade. Assim, havendo a necessidade de modelar sistemas complexos, onde se exige a adição de redundância em módulos do sistema, o usuário tem que recorrer a técnicas de modelagem hierárquica, fazendo uso, em conjunto, de modelos como CTMC e RBD, na tentativa de obter resultados mais expressivos.

2.3 Análise de Sensibilidade

A análise de sensibilidade é um método para determinar os fatores de maior influência em um sistema (FRANK, 1978; HAMBY, 1994). Os efeitos das mudanças na distribuição dos dados ou o impacto causado a partir de mudanças nos parâmetros são exemplos dos assuntos relacionados às análises de sensibilidade. De acordo com (HAMBY, 1994), alguns autores consideram que modelos são sensíveis aos parâmetros de entrada de duas diferentes formas: (1) a variabilidade, ou incerteza, de um parâmetro de entrada sensível tem uma grande contribuição à variabilidade de saída total, e (2) pode haver uma alta correlação entre um parâmetro de entrada e os resultados do modelo, o que significa dizer que pequenas mudanças nos valores de entrada resultam em mudanças significativas na saída. Esses dois tipos de sensibilidade paramétricas são manipulados através do uso de diferentes tipos de análises.

Quando relacionada com modelos analíticos como cadeias de Markov, a análise de sensibilidade paramétrica torna-se uma técnica particularmente importante para que os efeitos

causados através de mudanças nas taxas de transição nas métricas de interesse sejam computados. Essa abordagem pode ser utilizada para encontrar gargalos de desempenho ou confiabilidade no sistema, guiando assim a otimização dos processos (BLAKE; REIBMAN; TRIVEDI, 1988; ABDALLAH; HAMZA, 2002). Outro benefício da análise de sensibilidade é a identificação dos parâmetros que podem ser removidos sem efeitos significantes aos resultados. Modelos grandes, com uma dúzia de taxas, podem ser drasticamente reduzidos através do uso dessa abordagem.

Existem diversas formas de se conduzir uma análise de sensibilidade paramétrica. Algumas delas pode ser devidamente utiliza nos modelos analíticos como cadeias de Markov, visto que outras abordagens são mais adequadas para medições experimentais baseadas em análises. O método mais simples, numa visão conceitual, é o de variar repetidamente um parâmetro por vez, enquanto mantém os outros parâmetros sem variação (HAMBY, 1994). Quando aplicado este método, o *ranking* das sensibilidades é obtido a partir da observação das correspondentes mudanças na saída do modelo. Esse método é comumente utilizado em conjunção com parcelas de entrada *versus* os de saída. Tais parcelas permitem a detecção no gráfico de não-linearidades, não-monotonicidades, e de correlações entre as entradas e saídas dos modelos (MARINO et al., 2008). Relacionamentos inesperados entre variáveis de entrada e saída podem também ser reveladas a partir dessa abordagem, desencadeando a necessidade para outras investigações, baseando-se em diferentes abordagens (HAMBY, 1994). Uma determinada percentagem do valor médio do parâmetro pode ser utilizada para incrementar a abordagem citada. Cada parâmetro pode ser também incrementado a partir do fator de seu desvio padrão, quando essa informação é conhecida (DOWNING; GARDNER; HOFFMAN, 1985).

Contudo, variar um parâmetro por vez às vezes pode não ser a forma mais prática de realizar uma análise. Quando lidamos com um grande número de parâmetros, as análises das dispersões dos *plots* tornam-se mais difíceis, principalmente devido às proximidades das curvas. A diferença em termos de magnitude é outro possível fator crítico, uma vez que todos os parâmetros não podem ser visualizados no mesmo *plot*, inibindo interpretações precisas das diferenças entre as influências dos parâmetros. Devido a tais casos, métodos que são baseados em índices de sensibilidade numéricos devem ter preferências no uso da abordagem "um parâmetro por vez".

Outro método de análise de sensibilidade é o de análise diferencial. Com essa técnica, calcula-se a derivada parcial da métrica de interesse com relação a cada parâmetro de entrada (HAMBY, 1994). Por exemplo, considerando a métrica Y que depende do parâmetro (λ), a sensibilidade de Y com relação a λ é computada através das equações 2.16 ou 2.17:

$$S_{\lambda}(Y) = \frac{\partial Y}{\partial \lambda} \tag{2.16}$$

$$S^{*}_{\lambda}(Y) = \frac{\partial Y}{\partial \lambda} \left(\frac{\lambda}{Y} \right) \tag{2.17}$$

2.4 Injeção de Falhas

Técnicas de injeção de falhas são muito úteis para testar a resiliência de sistemas em geral (DALZIELL; MCMANUS, 2004). 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 et al., 2005). Com a utilização de injetores de falhas é permitido inserir tanto falhas no *software*, quanto no *hardware*. A injeção de falhas também pode ser utilizada como uma técnica de validação de sistemas.

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).

O método utilizado durante essa pesquisa para a obtenção da validação do modelo foi o proposto por (KEESEE, 1965), cuja finalidade é encontrar o intervalo de confiança da disponibilidade. Porém, para que esse valor seja alcançado, primeiro temos que obter dados como: tempo total de falhas e reparos, além da quantidade de eventos aplicados durate o processo de validação. A partir do número das ocorrências das falhas e dos reparos durante o processo de validação, podemos alcançar o grau de liberdade, que é a soma dessas ocorrências. Em seguida, o grau de liberdade é aplicado em uma função de distribuição-F através da ferramenta *Minitab* (MINITAB, 2015), para que os limites máximo e mínimo sejam alcançados a partir de um intervalo de confiança de 95%.

Através de *scripts* de injeção de falhas, é possível obter os valores de tempo total de falha e reparo durante o processo de validação. A variável ρ será a relação entre o tempo total de reparo e o tempo total de falha do sistema. Em posse do valor de ρ e dos limites máximo e mínimo encontrados através da distribuição-F, é possível alcançar os valores de ρ máximo (ρ_U) e ρ mínimo (ρ_L). O alcance desses valores também pode ser representado pelas seguintes Equações 2.18 e 2.19:

$$\rho L = \frac{\hat{\rho}}{f2n;2n;\frac{\alpha}{2}} \tag{2.18}$$

E ρ_U:

$$\rho U = \frac{\hat{\rho}}{f2n;2n;1-\frac{\alpha}{2}} \tag{2.19}$$

Por fim, depois de alcançar os valores de ρ, o estimador de probabilidade máximo para a disponibilidade é $\hat{A} = \frac{1}{1+\rho}$. Uma vez que a disponibilidade A é uma função monotonicamente decrescente de ρ, o intervalo de confiança 100*(1-α) para A, é dada pelas Equações 4.16 e 4.17 a sequir, que resulta no intervalo máximo (A_U) e mínimo (A_L) da disponibilidade:

$$A_U = \frac{1}{1+\rho U} \tag{2.20}$$

E D_L:

$$A_L = \frac{1}{1+\rho L} \tag{2.21}$$

2.5 Computação em Nuvem

A computação em nuvem é um modelo computacional cujos recursos como poder de processamento, rede, armazenamento e software são entregues como serviço através da Internet e podem ser acessados remotamente (ARMBRUST et al., 2010b). Esse modelo permite que usuários possam obter tais recursos de forma elástica, sob demanda e a um custo reduzido, entregues de forma semelhante a serviços tradicionais como água, gás, eletricidade e telefonia (DINH et al., 2013). A computação em nuvem fornece uma gama de informações em relação às quais os usuários não precisam se preocupar quanto à localização; precisam saber apenas que elas existem e que poderão acessá-las em qualquer lugar e a qualquer momento, desde que haja conexão com a rede.

Uma definição formal de computação em nuvem pode ser retirada do Instituto Nacional de Padrões e Tecnologia dos Estados Unidos da América (NIST - *National Institute of Standards and Technology*), onde afirma que computação em nuvem é um modelo que permite acesso ubíquo, conveniente, sob demanda a um *pool* compartilhado de recursos computacionais (i.e, redes, servidores, armazenamento, aplicativos e serviços) que podem ser rapidamente configurados e liberados com o mínimo esforço de gerenciamento ou interação com o provedor de serviços (STANDART; TECHNOLOGY'S, 2013).

Os avanços tecnológicos e a necessidade de utilizar grande poder de computação têm ajudado na utilização de tecnologias em nuvem. A computação em nuvem é importante para a distribuição e acesso de recursos de computação, oferecendo algumas vantagens de recursos computacionais (ARAUJO et al., 2011). A seguir, algumas características de nuvens:

• Escalabilidade e elasticidade: essa característica é a que dá a impressão de que os recursos computacionais da nuvem são infinitos, visto que são disponibilizados sob demanda (GOLDBERG, 1974). Para suportar as mais variadas demandas em diferentes momentos, algumas propriedades (como escalabilidade e elasticidade) são necessárias. A partir da escalabilidade, é possível adequar o sistema a um aumento de carga suportada, equivalente a quantidade de recursos (EUCALYPTUS CLOUD COMPUTING PLATFORM - ADMINISTRATOR GUIDE, 2010). O fornecimento e liberação dos recursos em tempo real ocorre devido à elasticidade. A união dessas duas características fortalece o modelo de computação em nuvem.

• Segurança: acessos realizados a aplicativos devem ser realizados apenas por pessoas autorizadas; usuários devem confiar seus dados à empresas que fornecem os serviços (CAROLAN et al., 2009b).

• Virtualização de recursos: é uma alternativa já bem consolidada em diversos cenários, como emulação de plataformas e ambientes que necessitam de maior desempenho computacional. Através 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 (MELO; MACIEL, 2014).

• Modelo de pagamento baseado em consumo: 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 (MELO; MACIEL, 2014).

2.5.1 Modelos de Negócio

Os serviços oferecidos na computação em nuvem envolvem plataforma de *hardware* e *software* sob demanda. Em geral, de acordo com o tipo de capacidade fornecida, os serviços de computação em nuvem são amplamente divididos em três categorias: Infraestrutura como um Serviço (*IaaS*), Plataforma como um Serviço (*PaaS*) e *Software* como um Serviço (*SaaS*) (GONG et al., 2010) (EUCALYPTUS, 2014a).

IaaS (infraestrutura como serviço) fornece uma infraestrutura de armazenamento, processamento e rede como serviço onde o cliente pagará apenas por aquilo que usar. Por exemplo, em um serviço de armazenamento, o usuário não pagará pelo disco completo, apenas pela capacidade que estiver consumindo. Empresas que disponibilizam esse tipo de serviço oferecem recursos computacionais na Internet por meio de máquinas virtuais. Exemplos de serviços, ou produtos, de *IaaS* são *Google Compute Engine* (GOOGLE, 2014a), *Amazon EC2* (EC2, 2014) e *GoGrid* (GOGRID, 2014).

PaaS (plataforma como serviço) fornece uma plataforma de desenvolvimento como serviço; suporta um conjunto de interface de aplicativos de programas para aplicações na nuvem. É a ponte intermediária entre *hardware* e aplicação. Com PaaS os usuários podem realizar ações como desenvolver, compilar, debugar, além de implantação e teste na nuvem. Exemplos de serviços são *Google App Engine* (ENGINE, 2014) e *Windows Azure* (MICROSOF, 2014).

SaaS (*Software* como serviço) fornece um conjunto de software como serviço na nuvem, visando a substituição daqueles antes instalados nos computadores pessoais. Usuários da nuvem podem liberar seus aplicativos em um ambiente de hospedagem que ode ser acessado na rede por diversos clientes *zhang2010cloud*. Com objetivo de vender *software* como serviço, o cliente pagará um valor relativamente menor do que o valor de compra do *software* em questão. Exemplos de tipos de serviços são *Google Drive* (GOOGLE, 2014b), *Sales Force* (FORCE, 2014) e serviços de *e-mail* como o *Gmail* (GOOGLE, 2014c).

2.5.2 Tipos de Nuvens

Como dito anteriormente, uma nuvem consiste basicamente em conjuntos de recursos computacionais mantidos e gerenciados por provedores, que disponibilizam tais recursos para seus clientes de acordo com a necessidade de cada um deles. Cada provedor tem suas particularidades na hora de oferecer o serviço da nuvem aos seus clientes. Enquanto que uns estão mais focados na redução de custo da operação, outros podem ter maior interesse em prover um serviço de maior confiabilidade e segurança de dados (ZHANG; CHENG; BOUTABA, 2010). Dessa forma podemos descrever os três tipos de nuvem (Figura 2.8), cada um com suas características, quanto à natureza de acesso e controle relacionadas ao uso e provisionamento de recursos físicos e virtuais.

Uma das formas de caracterizar uma nuvem computacional é a partir do ponto de vista do usuário da nuvem e de quem mantém a infraestrutura. Nesse ponto, podemos classificar uma nuvem através de três tipos (ZHANG; CHENG; BOUTABA, 2010):

Privada: são infraestruturas de nuvem projetadas para uso de uma única organização, cujo gerenciamento pode ser realizado pela própria empresa ou por um terceiro. Normalmente as empresas que optam pelo gerenciamento da própria nuvem são organizações que operam dados críticos, como é o caso de bancos, órgãos do governo, agências militares, etc. Por outro lado, esse tipo de infraestrutura requer um investimento inicial muito alto em uma *server farm* proprietária (ZHANG; CHENG; BOUTABA, 2010), de forma igual ao modelo tradicional sem o uso de nuvem. Por outro lado, observa-se as vantagens na consolidação e gerenciamento dos recursos (ZHAO; FIGUEIREDO, 2007; KIM; MACHIDA; TRIVEDI, 2009);

Pública: são infraestruturas mantidas por provedores, que oferecem seus recursos físicos para outras organizações e usuários comuns. Diferente da nuvem privada, o custo inicial de infraestrutura de nuvem pública por parte do usuário é zero, visto que a aquisição de servidores e outros equipamentos acontece pela organização que está provendo o serviço. Uma das vantagens da nuvem pública é que os recursos são fornecidos dinamicamente, ou seja, caso haja um aumento na necessidade de recursos virtualizados, a oferta se ajusta à demanda, e a responsabilidade de gerenciamento de infraestrutura é transferida para o provedor da nuvem (CAROLAN et al., 2009b). Por outro lado, esse tipo de nuvem tem como desvantagem a falta de controle sobre os dados, rede e configurações de segurança, visto que os dados de milhares usuários podem estar alocados em uma única nuvem. Tal desvantagem é o que impede algumas organizações de utilizar esse tipo de nuvem, principalmente as que lidam com dados críticos.

Híbrida: esse modelo representa uma combinação entre a nuvem pública e a privada, onde tanto a organização provedora do serviço quanto o usuário são responsáveis por manter a infraestrutuara em funcionamento. A nuvem híbrida oferece maior flexibilidade que os demais tipos de nuvens, além de prover maior controle e segurança sobre os dados e aplicações em comparação às nuvens públicas, e facilitar a expansão sob demanda e contratação de serviços. Por outro lado, a complexidade no gerenciamento da nuvem também aumenta, e determinar o

particionamento ótimo entre os componentes públicos e privados pode ser um desafio (ZHANG; CHENG; BOUTABA, 2010).

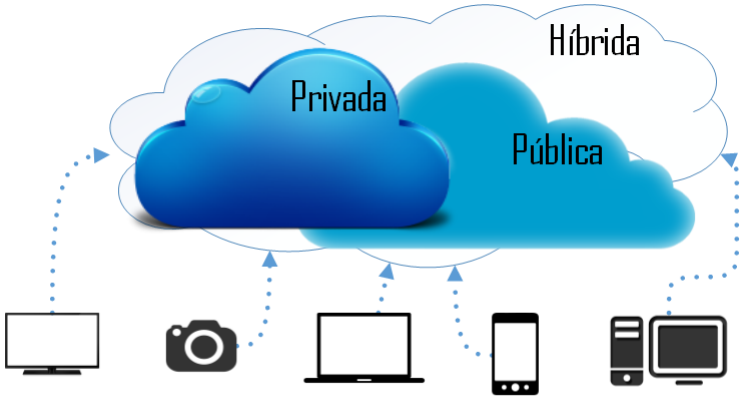


Figura 2.8: Tipos de Nuvens.

Além destes principais modelos descritos acima, também existe o modelo de *Data as a Service (DaaS)* (TRUONG; DUSTDAR, 2009), no qual a nuvem oferece ao seus clientes um serviço de armazenamento secundário, garantindo alta disponibilidade e integridade dos dados. Um serviço de DaaS pode ser visto como *IaaS* ou como *SaaS*, dependendo do contexto sob o qual é utilizado. Considere como exemplo o serviço de armazenamento e *backup* em nuvem do *Dropbox* (DROPBOX, 2014). O serviço que o *Dropbox* oferece para seus usuários pode ser enxergado como *SaaS* ou *DaaS*. Contudo, o próprio *Dropbox* utiliza o serviço de armazenamento da *Amazon* - S3 (*Simple Storage*) (SERVICE, 2014) - para armazenar os dados dos seus clientes. Do ponto de vista do *Dropbox*, o serviço de armazenamento da *Amazon* pode ser considerado como *IaaS*.

2.6 Plafatorma *Eucalyptus*

EUCALYPTUS - Elastic Utility Computing Architecture Linking Your Programs To Useful Systems - é um *software* que implementa estilos escaláveis de *IaaS* de nuvens privadas e híbridas (AMAZON, 2014; EUCALYPTUS, 2014c). O *Eucalyptus* foi criado com a finalidade de pesquisa em computação em nuvem e a interface compatível com o serviço comercial da *Amazon*, o *Amazon EC2* (EUCALYPTUS, 2014a). A compatibilidade da *API* permite executar um aplicativo da *Amazon* no *Eucalyptus* sem modificação. Em geral, a plataforma do *Eucalyptus* utiliza a capacidade de virtualização (*hypervisor*) do sistema de computador subjacente para permitir alocação flexível de recursos e computação dissociados de *hardware* específico (AMAZON, 2014; EUCALYPTUS, 2014c). *Eucalyptus* é compatível com pacotes para múltiplas distribuições do *Linux*, incluindo o *Ubuntu*, *RHEL*, *OpenSuse*, *Debian*, *Fedora*, e o *CentOS*.

A arquitetura do *Eucalyptus*, representada pela Figura 2.9, é composta por cinco componentes de alto nível, cada um com seu próprio *web service*: Controlador da Nuvem, Controlador

do *Cluster*, Controlador do Nó, Controlador de Armazenamento e *Walrus*. Segue uma descrição a respeito de cada um desses componentes.

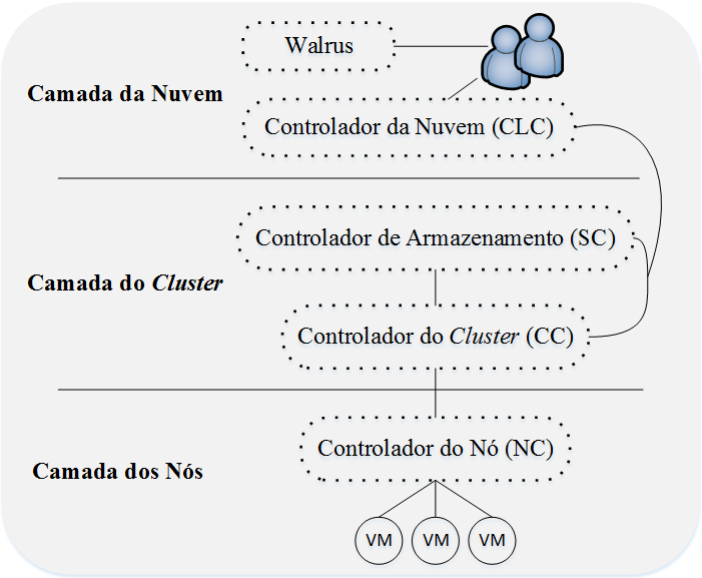


Figura 2.9: Arquitetura *Eucalyptus*.

2.6.1 Controlador Geral da Nuvem (GC)

O Controlador da Nuvem *Cloud Controller (CLC)*, é o *front-end* (ponto necessário para acesso de clientes na nuvem) para a infraestrutura de nuvem inteira. O CLC é responsável por expor e administrar os recursos subjacentes virtualizados (servidores, rede e armazenamento) via *API Amazon EC2* (EC2, 2014). Este componente utiliza interfaces de serviços da Internet para receber os pedidos de ferramentas de clientes de um lado e de interagir com o resto dos componentes do *Eucalyptus* no outro lado. Sua função é descrita a seguir (JOHNSON et al., 2010):

- Monitorar a disponibilidade de recursos em diversos componentes da infraestrutura de nuvem, incluindo o *hypervisor* no nó, que é utilizado para o provisionamento de recursos para as instâncias; e os controladores de *cluster*, que gerenciam o *hypervisor* no nó;
- Decidir quais *clusters* serão utilizados para o provisionamento das instâncias;
- Monitoramento das instâncias em execução.

2.6.2 Controlador do *Cluster* (CC)

O Controlador de *Cluster* (*Cluster Controller (CC)*) geralmente é executado em uma máquina que esteja conectada às máquinas dos NCs e CLCs em execução. Além disso, os CCs reúnem informações a respeito do conjunto de máquinas virtuais e seus horários de execução nos

NCs específicos. Suas principais funções são: agendar visitas, implantação e execução para NCs específico; controlar a instância virtual de rede overlay e reunir/relatar informações sobre um conjunto de NCs (EUCALYPTUS, 2014c). Assim, o CC deverá conferir os recursos disponíveis informados por cada Controlador de Nó e decidir qual máquina física deverá instanciar as *VMs* requisitadas pelo CLC. O CC comunica com Controlador da Nuvem (CLC) de um lado e NCs, do outro lado. As funções desse componente são divididas da seguinte maneira (JOHNSON et al., 2010):

- Receber pedidos do CLC para implantar instâncias;
- Decidir qual NCs usar para implantar as instâncias;
- Controlar a rede virtual disponível para as instâncias;
- Coletar informações sobre os NCs registrados e comunicar o fato ao CLC.

2.6.3 Controlador do Nó (NC)

O Controlador do Nó (*Node Controller (NC)*) é executado em cada nó físico e controla o ciclo de vida das instâncias em execução no nó. O NC interage com o sistema operacional e com o *hypervisor* em execução no nó. Os NCs controlam a execução, fiscalização e terminação das instâncias de *VMs* no *host* onde está sendo executado. Ele consulta e controla o *software* do sistema em seu nó, em resposta às consultas e solicitações do controle da CC (EUCALYPTUS, 2014c). O NC faz consultas para descobrir recursos físicos do nó - o número de núcleos de CPU, tamanho da memória, espaço em disco disponível - assim como para aprender sobre o estado das instâncias *VM* nesse nó (JOHNSON et al., 2010). Sendo assim, podemos dividir as funções do NC da seguinte forma:

- Coletar dados relacionados com a disponibilidade e utilização de recursos no nó e apresentar os dados para o CC;
- Ciclo de vida das instâncias.

2.6.4 Controlador do Armazenamento (SC)

O Controlador de armazenamento (*Storage Controller (SC)*) fornece armazenamento de bloco persistente para uso pelas instâncias de máquinas virtuais. Ele implementa acesso a bloco de armazenamento em rede, semelhante ao proporcionado pela *Amazon Elastic Block Storage - EBS* (EUCALYPTUS, 2014c), e é capaz de interagir com vários sistemas de armazenamento (NFS, iSCSI, etc.). Um *Elastic Block Storage* é um dispositivo de bloco *Linux* que pode ser ligado a uma máquina virtual, mas envia o tráfego de disco em toda a rede ligada localmente para um local de armazenamento remoto. Um volume EBS não pode ser compartilhado entre

instâncias (EUCALYPTUS, 2014c). A sua função é dividida desse modo (JOHNSON et al., 2010):

- Criar dispositivos persistentes EBS;
- Proporcionar o armazenamento de bloco com os protocolos AoE ou iSCSI para as instâncias;
- Permitir a criação de volumes instantâneos.

2.6.5 Walrus

O *Walrus* é um arquivo baseado em serviço de armazenamento de dados, sendo sua interface compatível com a *Amazon's Simple Storage Service* (S3) (JOHNSON et al., 2010). O *Walrus* implementa uma interface REST (através de HTTP), às vezes chamada de interface "*Query*", bem como interfaces SOAP, que são compatíveis com o S3 (JOHNSON et al., 2010), (EUCALYPTUS, 2014c). Os usuários que têm acesso ao *Eucalyptus* podem usar o *Walrus* para transmitir dados dentro/fora da nuvem, bem como instâncias onde eles tenham sido iniciados nos nós. Os Nós da arquitetura podem enviar imagens de *VMs* para o *Walrus*, bem como baixá-las para poder instanciar as *VMs*. Sua função é subdividida em três partes (JOHNSON et al., 2010):

- Armazenar as imagens de máquinas virtuais;
- Armazenamento instantâneo;
- Armazenar e servir arquivos usando API S3.

3

Plataforma de Vídeo *Streaming*

Devido ao crescente uso da Internet no mundo todo, do uso dos smartphones e outras tecnologias, tem-se observado um aumento na demanda por informações multimídias pela Internet, principalmente no que diz respeito ao vídeo *streaming*. Para esse tipo de serviço, é necessário que haja uma rede confiável para evitar que ocorram problemas com, por exemplo: largura de banda, *delay*, diminuição da qualidade do conteúdo e perda de pacotes. É necessário também que o servidor do serviço de *streaming* faça uso de soluções que evitem ou amenizem esse impactos de *QoS* diante de instabilidades da rede, adequando, por exemplo, o tratamento da mídia a ser transmitida de acordo com o *status* da rede do cliente/receptor.

Este capítulo tem como finalidade apresentar a composição de uma arquitetura de *streaming* de mídias, apontando e trazendo uma breve descrição das seis áreas que formam uma arquitetura para o provimento de um serviço *streaming*. Em seguida, apresentamos o serviço de vídeo *streaming* que foi elaborado para realizar esta pesquisa.

3.1 Arquitetura para Vídeo *Streaming*

A Figura 3.1 apresenta uma arquitetura de video *streaming*. Nessa representação, o vídeo bruto é pré-comprimido por um algoritmo de **compressão de vídeo**, e então é salvo em um dispositivo de armazenamento. A partir de uma requisição de um usuário, um **servidor de *streaming*** recupera o arquivo do vídeo comprimido no dispositivo de armazenamento. Em seguida, o módulo **camada de aplicação de controle de *QoS*** adapta os bits de streaming do arquivo de mídia de acodo com o status da rede e requerimentos de *QoS* do usuário. Depois dessa adaptação, os **protocolos** de transporte empacotam os bits de streaming e enviam os pacotes do conteúdo de mídia para a Internet. Esses pacotes podem se perder ou passar por um *delay* pela Internet devido a algum tipo de congestionamento. Para melhorar a qualidade da transmissão da mídia, soluções como **serviço de distribuição de mídia contínua** (por exemplo por cache) são adotados. Para os pacotes que são entregues com sucesso seguem as seguintes etapas: primeiramente passam pela camada de transporte e então são processados pela camada de aplicação, antes de serem decodificados no decodificador da mídia. Para que haja sincronização

entre as apresentações de áudio e vídeo, é preciso utilizar um **mecanismo de sincronização de mídia**. A seguir, representamos graficamente o fluxo dos tópicos destacados, bem como uma breve descrição de cada um deles.

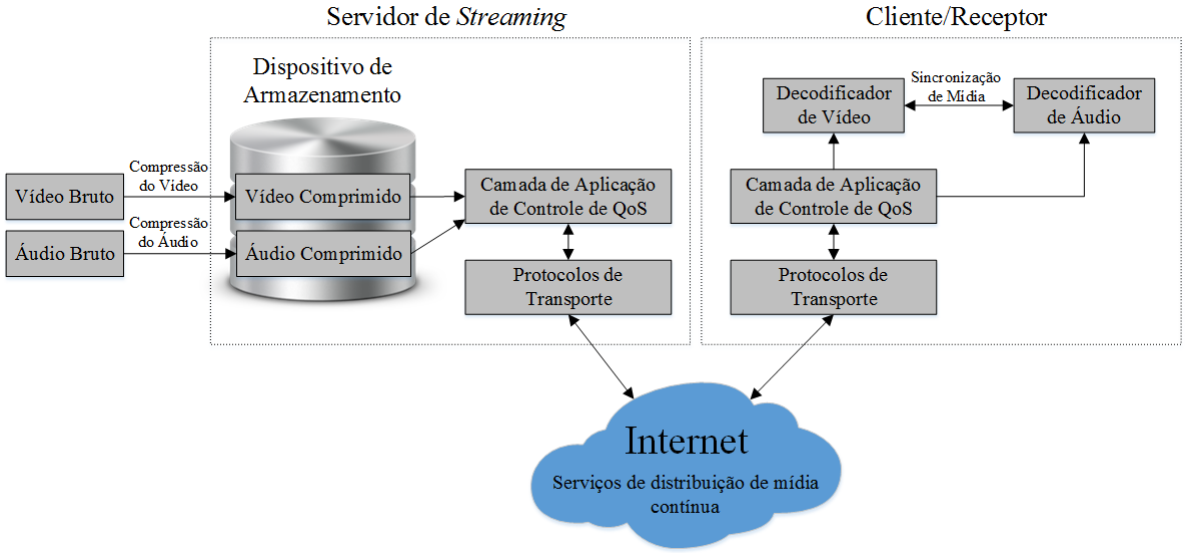


Figura 3.1: Arquitetura para Vídeo *Streming*.

Compressão de vídeo: o vídeo bruto deve ser comprimido antes da transmissão. O plano de compressão de vídeo pode ser classificado em duas categorias: codificação de vídeo escalável e não-escalável. A categoria de vídeo escalável é capaz de lidar com as oscilações de largura de banda da Internet (MCCANNE; JACOBSON; VETTERLI, 1996), que são os itens de maior preocupação no que se diz respeito às técnicas de codificação de vídeo escalável.

Camada de aplicação de controle de *QoS*: para lidar com situações de variações de rede e diferenças na qualidade de apresentação das requisições feitas pelos usuários, algumas técnicas de controle de *QoS* da camada de aplicação foram propostas por (ELEFTHERIADIS; ANASTASSIOU, 1995), (TAN; ZAKHOR, 1999), (XIN; SCHULZRINNE, 1999), (ZHANG et al., 2001). As técnicas de camada de aplicação incluem controle de congestionamento e de erro. As respectivas funcões dessas técnicas são: é aplicado para previnir perdas de pacotes e reduzir o *delay*. O controle de erro, por outro lado, tem como função melhorar a apresentação da qualidade do vídeo diante de perdas de pacotes. Dentre os mecanismos de controle de erro, encontramos o de correção de erro adiante, retransmissão, codificação de erro resiliente e dissimulação de erro.

Serviço distribuído de mídia contínua: a fim de prover apresentações multimídias com qualidade, é crucial que se tenha um suporte de rede adequado com o objetivo de reduzir o *delay* e as perdas de pacote. Através do serviço distribuído de mídia contínua, é possível alcançar *QoS* e eficácia na realização do *streaming* de vídeo/áudio sobre o melhor-esforço pela Internet. Serviço distribuído de mídia contínua também inclui filtragem de rede, nível de aplicação *multicast* e replicação de conteúdo.

Servidor de streaming: servidores de *streaming* representam o ponto chave do provimento do serviço de *streaming*. Para que um serviço de *streaming* de qualidade seja ofertado, é

necessário um servidor de *streaming* para processar os dados multimídia considerando restrições de tempo e suporte para operações de controle interativo como *pause/resume*, adiantar/atrasar conteúdo, etc. Além disso, servidores de *streaming* precisam recuperar componentes de mídia de forma sincronizada. Um servidor de *streaming* é composto tipicamente por três subsistemas, que são chamados de: um comunicador (por exemplo: protocolos de transporte), um sistema operacional e um sistema de armazenamento.

Mecanismos de sincronização de mídia: sincronização de mídia é a maior característica que diferencia aplicações multimídia de quaisquer outras aplicações tradicionais de dados. Com esses tipos de mecanismos, a aplicação no lado do receptor pode apresentar diversos streams de mídia da mesma maneira que eles foram originalmente capturados. Um exemplo de sincronização de mídia é que os movimentos dos lábios de alguém que está falando, condiz com a saída do áudio que está sendo transmitido.

Protocolos para streaming de mídia: protocolos são projetados e padronizados para que haja comunicação entre os clientes e os servidores de *streaming*. Protocolos para *streaming* de mídia tem a finalidade de prover tanto serviços quanto endereçamento de rede, transporte e sessões de controle. De acordo com cada funcionalidade, os protocolos podem ser classificados em três categorias: protocolo de camada de rede (como *Internet Protocol (IP)*), protocolo de transporte (como *User Datagram Protocol (UDP)*) e protocolo de controle de sessão como o protocolo de streaming em tempo real (*Real-time Streaming Protocol (RTSP)*).

3.2 Serviço de *VoD streaming*

Como foi visto, o *streaming* é a transmissão de tipos diversos de mídias (áudio, vídeo ou imagens) através da rede, enquanto que o consumo dessas mídias acontece simultaneamente. O provimento desse tipo de serviço não se restringe apenas a grandes provedores, pois com o uso de ferramentas gratuitas como o *VLC media player* aliado a *IaaS* da computação em nuvem, é possível criar um ambiente de serviço de *VoD streaming* de forma satisfatória. Dessa forma, essa subseção apresenta o serviço de vídeo *streaming* que foi elaborado para esta pesquisa, apresentando os componentes que formam a arquitetura mais básica, as aplicações que foram utilizadas e as configurações disponíveis.

A Figura 3.2 representa o funcionamento do serviço de *VoD streaming* a partir da arquitetura básica do serviço, com base na plataforma *Eucalyptus*. Essa arquitetura inicial é composta por três componentes: o usuário, o serviço em nuvem e uma conexão via *swicth* entre estes dois. Mais adiante, a Seção 4.2.1 apresenta a arquitetura geral (em termos de estrutura) e os modelos de disponibilidade para esta arquitetura básica, bem como os detalhes de cada componente da arquitetura, que influencia na disponibilidade do sistema.

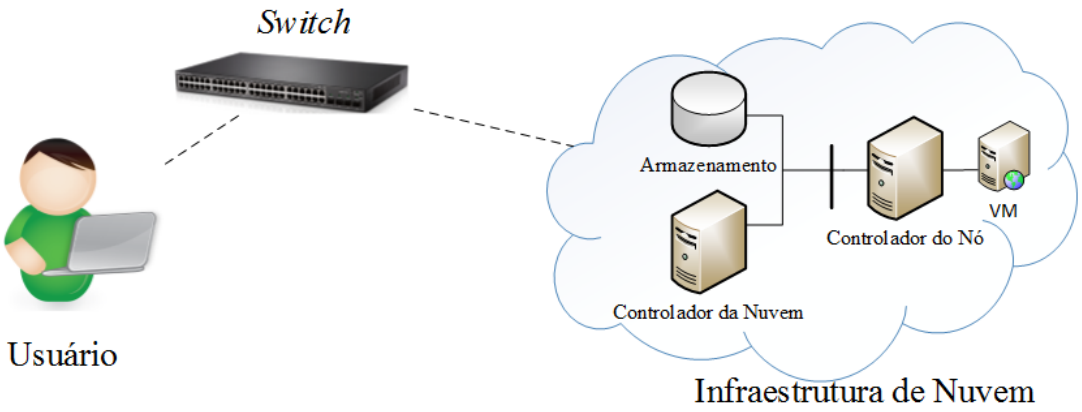


Figura 3.2: Arquitetura Lógica do Sistema.

O provimento do serviço inicia-se a partir da requisição do *streaming* pelo usuário através do *player* da aplicação *VLC*, que deve estar em conexão com a rede. A infraestrutura da nuvem é composta pelos seguintes componentes: GC, armazenamento, NC e a *VM*. O GC e o NC são componentes da plataforma *Eucalyptus*, apresentados anteriormente na Seção 2.6. O armazenamento representa o dispositivo externo *"iomega ix4"*, cuja capacidade de armazenamento é de 8 *TB*, podendo armazenar e transmitir mídias pela rede. É nele que os vídeos do serviço são armazenados.

O servidor do serviço de *streaming* é hospedado na *VM* do ambiente da nuvem, e é formado pelas aplicações *Apache* e *VLC*. A função do *Apache* é permitir que o serviço seja transmitido pela Internet, enquanto que a função do *VLC* é dar o suporte para que o *streaming* aconteça e seja transmitido até o usuário. O *VLC* foi escolhido para essa pesquisa porque ele suporta um grande número de formato de vídeos, e por ser uma ferramenta gratuita. Essa *VM* é instanciada através de uma imagem personalizada criada a partir do *Virt-Manager* (MANAGER, 2014), e tem o *Ubuntu Server* (UBUNTU, 2014) como sistema operacional, trazendo as aplicações *Apache* e *VLC* já instaladas. Foi necessário personalizar a imagem da *VM* para agilizar e automatizar o processo de reparo do serviço em uma situação de falha. Dependendo do tamanho da *VM*, varia-se o número requisições de *streaming* que podem ocorrer enquanto o sistema estiver disponível.

Assim, o serviço de *VoD streaming* se inicia a partir da solicitação de uma transmissão pelo usuário. Essa solicitação é encaminhada ao servidor de *VoD*, hospedado na *VM*, que por sua vez busca a mídia codificada no Armazenamento. Se a mídia for encontrada, o *VoD server* retorna a requisição ao usuário, que terá o conteúdo decodificado pelo *plugin* do *VLC*. Caso a requisição não seja encontrada no Armazenamento, nada é retornado ao usuário.

O serviço de vídeo *streaming* foi testado em quatro tipos diferentes de *VMs* do *Eucalyptus*, que correspondem às instâncias padrões do *Amazon EC2* (EC2, 2014). As especificações dessas *VMs* estão descritas na Tabela 3.1. Os *scripts* utilizados durante esse estudo estão listados nos apêndices dessa dissertação.

As configurações de *hardware* das máquinas utilizadas nesse serviço de *VoD streaming*

Tabela 3.1: Especificações das Máquinas Virtuais.

Tipo	CPU	Memória (MB)	HD (GB)
m1.Small	1	256	5
m1.Medium	1	512	10
m1.Large	2	512	10
m1.Xlarge	2	1024	10

são dadas pela Tabela 3.2. Com relação ao *software* utilizado, as máquinas do ambiente da nuvem utilizaram o *Eucalyptus* versão 3.2.2 e o sistema operacional *CentOS* (CENTOS, 2014) versão 6.4. A máquina utilizada como usuário utiliza o *Ubuntu Server Linux* versão 12.04, e é utilizada apenas para requisitar vídeos ao serviço, não sendo necessário fazer uso de uma configuração robusta.

Tabela 3.2: Configurações de *hardware* das máquinas utilizadas.

Recursos	GC	NC	Usuário
Processador	Intel Xeon E3	Xeon E5	Core i7
Memória (GB)	8	8	8
HD (GB)	500	1000	1000

4

Metodologia e Modelos

A necessidade de promover serviços de computação em nuvem mais confiáveis, disponíveis e seguros, favorece o desenvolvimento e uso de técnicas, estratégias e modelos de avaliação de disponibilidade para tais serviços, uma vez que, mesmo contando com um grande provisionamento de *IaaS*, esses sistemas não estão isentos de eventos de falha de *hardware* e *software*. Este capítulo apresenta a metodologia utilizada para avaliação de disponibilidade de um serviço de *VoD streaming* na nuvem, e também apresenta as arquiteturas desenvolvidas e avaliadas, bem como os modelos elaborados para avaliação de disponibilidade dessas arquiteturas, cujo funcionamento foi apresentado na Seção 3.2.

Primeiro, a metodologia utilizada para a concepção desse trabalho é descrita, partindo de uma arquitetura básica e, através de modelagem por RBD e CTMC, alcança as métricas de análise de disponibilidade. Em posse desses resultados, novas arquiteturas são propostas a fim de obter melhores resultados de disponibilidade. Na sequência, são apresentados os modelos da arquitetura básica, cuja apresentação é realizada na subseção 4.2.1, e seus componentes são detalhados. Em seguida, a partir da identificação de gargalos de disponibilidade, variações da arquitetura básica são apresentadas nas subseções 4.2.2, 4.2.3 e 4.2.4, com o objetivo de alcançar melhores resultados de disponibilidade do serviço de *VoD streaming*. As duas variações das arquiteturas são elaboradas a partir de componentes redundantes, e são classificadas como: arquitetura redundante - NC, e arquitetura redundante - NC e GC. O mecanismo de replicação utilizado para cada uma das arquiteturas redundantes é o *warm-stand*, que é ativado apenas quando a unidade em operação falha.

A partir dos modelos alcançados, é importante que os resultados obtidos estejam condizentes com o comportamento do sistema real. Para isso, foi realizado a validação do modelo que representa o funcionamento do serviço de *VoD streaming*, na subseção 4.3.

4.1 Metodologia

A metodologia aplicada para realizar a avaliação de disponibilidade das arquiteturas do serviço de *VoD streaming* na nuvem consiste no cumprimento de uma sequência de passos como

representa a Figura 4.1. Cada um dos passos é detalhado nas subseções que seguem.

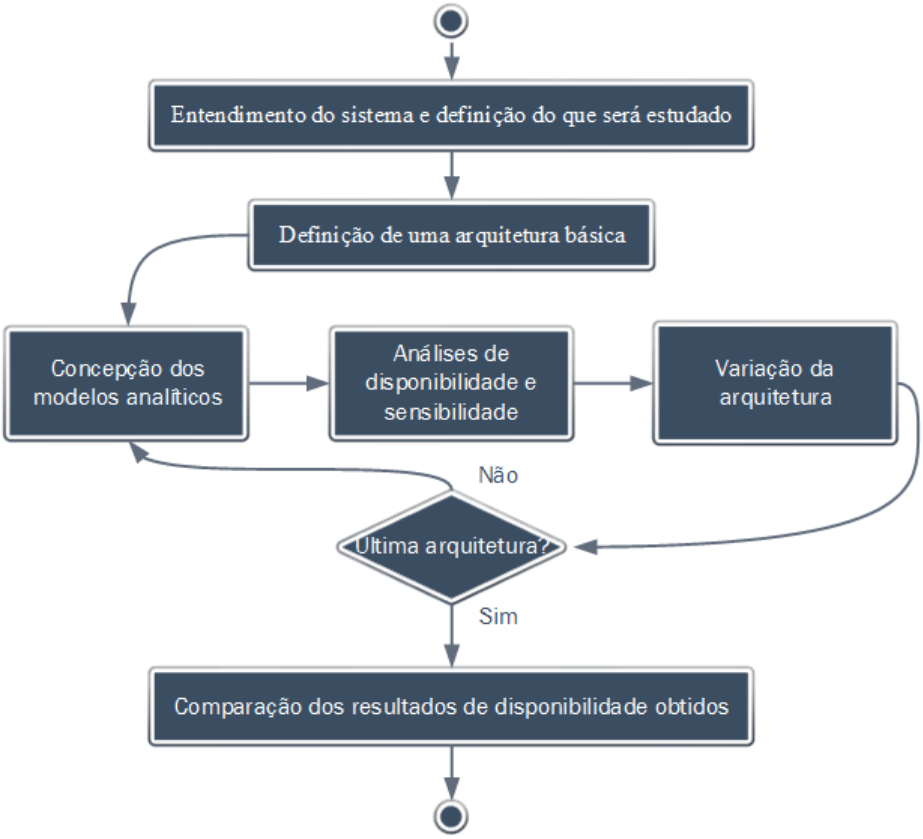


Figura 4.1: Fluxo do desenvolvimento.

- **Entendimento do sistema e definição do que será estudado:** essa primeira fase refere-se ao entendimento do sistema, seus componentes, aplicações e funcionalidades. Nesse ponto é importante a identificação do que será analisado durante a pesquisa, definindo os requisitos de disponibilidade que serão considerados no processo de avaliação do serviço.

  Como já foi mencionado, nosso trabalho tem como objetivo observar resultados de disponibilidade para um serviço de *VoD streaming* a partir de arquiteturas baseadas na infraestrutura de uma nuvem privada. Nessa fase inicial, encontramos diversos estudos relacionados à análise de disponibilidade de nuvens privadas, mas que não abordavam a inclusão de serviços nessas arquiteturas. Sobretudo, não identificamos a exploração de serviços de *VoD streaming* nesse contexto, já que é um tipo de serviço que demanda elasticidade e alta disponibilidade. Diante disso, nos preocupamos em iniciar o planejamento da pesquisa a partir de infra-estruturas que proporcionassem elasticidade de recursos, bem como alta disponibilidade. Portanto, tivemos que definir qual a plataforma de nuvem privada seria mais adequada, o tipo de *streaming* que poderíamos analisar, quais as aplicações que dariam suporte ao provimento do serviço, o número de usuários suportados, bem como o consumo de banda por

usuário. Além disso, como utilizaríamos *VMs* para o serviço, consideramos os tipos de *VMs* e suas configurações.

Com o entendimento da fase inicial, foi possível estabelecer que a plataforma *Eucalyptus* seria a adotada nesse estudo, devido a sua vasta utilização em contextos de computação em nuvem. Com isso, a arquitetura básica foi definida, bem como os componentes que iriam compor essa arquitetura inicial. Além disso, diante das ferramentas que possibilitam a transmissão de conteúdo via *streaming*, optamos por utilizar a ferramenta *VLC* (VIDEOLAN, 2014), visto que é uma ferramenta sem custos de aquisição, e que é capaz de executar diversos tipos de formatos de vídeos. Como suporte para o *VLC*, optamos pelo *Apache* (APACHE, 2014), que é um *web service* gratuito e amplamente utilizado para estes fins.

- **Definição de uma arquitetura básica:** diante da definição do que seria estudado, bem como da escolha das ferramentas que seriam utilizadas para prover o serviço definido, a primeira arquitetura é elaborada. Essa arquitetura é a primieira, portanto a mais básica. A partir dessa proposta inicial é que novas arquiteturas serão propostas após o cumprimento das etapas desta metodologia.

- **Concepção dos modelos analíticos:** a partir da definição da arquitetura básica e dos componentes que formam tal arquitetura, tivemos que elaborar modelos analíticos para representar o funcionamento da infraestrutura de nuvem proposta. Para isso, foi necessário investigar qual o tipo de modelagem seria a mais adequada para o nosso estudo.

  Para representar o relacionamento entre os componentes que formam nossa arquitetura, fizemos uso de diagramas de blocos de confiabilidade: RBD. Como o funcionamento do serviço do bloco que representa o serviço reunia interações entre a *VM* e as aplicações necessárias, fizemos uso do modelo combinatório baseado em estados: CTMC. Para cada modelo, seja na arquitetura básica, seja nas variações das arquiteturas propostas, foi necessário buscar valores de falha e reparo de cada componente dos modelos propostos. Muitos desses valores foram encontrados em artigos e *journals* correlatos a esta pesquisa. Porém, alguns outros valores foram obtidos a partir de coletas de tempos e experimentos realizados por nós, durante a pesquisa.

  A partir da definição dos tipos dos modelos analíticos, bem como o alcance das métricas de falha e reparo de cada um dos componentes que formam a arquitetura proposta, os modelos analíticos são avaliados e resultados de disponibilidade, como disponibilidade e indisponibilidade anual, são obtidos.

- **Análises de disponibilidade e de sensibilidade:** A partir da definição da arquitetura a ser analisada, e do alcance de seus respectivos modelos analíticos, é possível

que haja o análise de disponibilidade da arquitetura, calculando-se os valores de disponibilidade e indisponibilidade anual - que são as métricas de interesse deste estudo, através dos valores de falha e reparo dos componentes que formam essa arquitetura inicial. Realizada a análide de disponibilidade da arquitetura, é preciso analisar uma nova arquitetura para observar o comportamento dos resultados de disponibilidade. Para que uma nova arquitetura seja proposta, este estudo considerou a técnica de análise de sensibilidade paramétrica para identificação do componente de maior criticidade na arquitetura básica, a fim de obter melhorias nos resultados de disponibilidade do serviço proposto.

- **Variação da arquitetura:** a partir da arquitetura obtida, a técnica de análise de sensibilidade paramétrica é aplicada com o objetivo de identificar índices de sensibilidade para os parâmetros dos modelos analisados. Como resultado da aplicação dessa técnica, um *raking* com as sensibilidades indica os gargalos de disponibilidade no modelo proposto, sugerindo que mudanças a partir do componente indicado pela sensibilidade possa trazer ganhos de disponibilidade e desempenho ao contexto analisado.

  Dessa forma, todas as arquiteturas propostas nesse trabalho foram elaboradas a partir dos gargalos identificados nos resultados das análises de sensibilidade realizadas. A utilização dessa técnica se justifica devido ao foco do trabalho, que é a de observar os resultados de disponibilidade para os cenários propostos.

- **Última arquitetura?:** diante dos resultos das análises de sensibilidade e disponibilidade da arquitetura analisada, observamos se os resultados são satisfatórios e com isso se mais uma arquitetura deverá ser analisa ou se os resultados indicados foram razoáveis. Caso o resultado seja satisfatório, o passo seguinte é a comparação dos resultados de disponibilidade obtidos. Caso o valor não seja satisfatório, mais uma vez inicia-se o processo de elaboração de modelos analíticos para uma nova arquitetura, bem como as análises de disponibilidade e sensibilidade para a proposta.

- **Validação do modelo analítico:** em posse dos modelos analíticos obtidos, é preciso que haja a validação de tais modelos, com o objetivo de garantir que eles apresentam o mesmo comportamento que o sistema real. Para o processo de validação, combinamos técnicas de injeção de falhas para obtenção de valores de falha e reparo do serviço, em seguida alcançamos o intervalo de confiança desses valores, para medir a confiança os resultados de disponibilidade alcançados. A injeção de falhas ocorreu através de scripts de *monitoramento* para cada um dos componentes do serviço: *Apache*, *VLC* e *VM*.

- **Comparação dos resultados de disponibilidade obtidos:** com os modelos analíticos alcançados, é possível realizar a análise de disponibilidade para que os valores

de disponibilidade sejam obtidos para cada um dos modelos alcançados. Dessa forma, com os resultados de disponibilidade para cada uma das arquiteturas propostas (básica e suas variações), é possível que haja a comparação entre as arquiteturas.

4.2 Modelos Analíticos

Esta Seção apresenta a arquitetura básica e suas variações, bem como seus respectivos modelos analíticos. A representação das arquiteturas propostas através de modelagem analítica acontece por meio de RBD e CTMC. A adoção de RBD é devido a esse tipo de modelagem indicar como que o funcionamento dos componentes que formam um sistema pode afetar o funcionamento de tal sistema. Além disso, RBD possibilita o cálculo de métricas de dependabilidade, como a disponibilidade (MACIEL et al., 2011), que é o objetivo desse estudo. Alguns componentes dos modelos RBD são formados por outros componentes, onde alguns podem ser representandos por outros RBDs, enquanto que outros apresentam interações mais complicadas entre seus componentes, justificando o uso de modelos baseados em estados em tais situações, como é o caso do CTMC (MACIEL et al., 2011). Esse tipo de modelo também é amplamente utilizado em análises de disponibilidade, além de tornar possível a obtenção de equações de fórmula fechada para obtenção de disponibilidade do que está sendo modelado. A ferramenta utilizada para modelagem dos modelos aqui propostos, é descrita no Apêndice A.

4.2.1 Arquitetura Básica

A arquitetura de infraestrutura básica desse estudo tem funcionamento idêntico ao apresentado na Seção 3. A composição dessa infraestrutura para o provimento do serviço de *VoD streaming* é dada por duas máquinas, sendo uma para o NC e outra para o GC, como indica a Figura 4.2. O NC irá disponibilizar recursos para que o *VoD Server* seja iniciado através da instância de uma VM. O GC irá gerenciar o ambiente de nuvem. As mídias do serviço estão armazenadas no *Volume*. A requisição do conteúdo de *streaming* é feita pelo usuário, através do *plugin* da aplicação *VLC*.

Com relação ao modo operacional dessa arquitetura, compreende-se que o serviço estará indisponível diante da ocorrência de falha em um dos seguintes componentes: NC, GC, Volume ou Serviço (*VM* e as aplicações Apache e VLC). Diante de uma falha no NC, o funcionamento do serviço é interrompido, visto que a *VM* (onde o VoD server está hospedado) é instanciada a partir dos recursos do NC. De forma similar, uma falha no GC impossibilita a prestação do serviço já que a gerência da nuvem estará comprometida. Diante da falha do componente Volume, as mídias não poderão ser transmitidas e o conteúdo de streaming estará indisponível. A falha da *VM* ou das aplicações que formam a *VM* - cuja integração está no bloco Serviço, interrompe o funcionamento do serviço, tornando-o indisponível. Portanto, consideramos que o serviço está disponível quando todos os seus componentes estão operantes.

Desta forma, o modo operacional para representar a plataforma do serviço de *VoD streaming* é representado pela expressão lógica $VoDService_{up}$ 4.1, onde Serviço compreende tanto a *VM* quanto as aplicações *Apache* e *VLC*.

$$VoDService_{up} = GC \wedge NC \wedge Volume \wedge Servidor \tag{4.1}$$

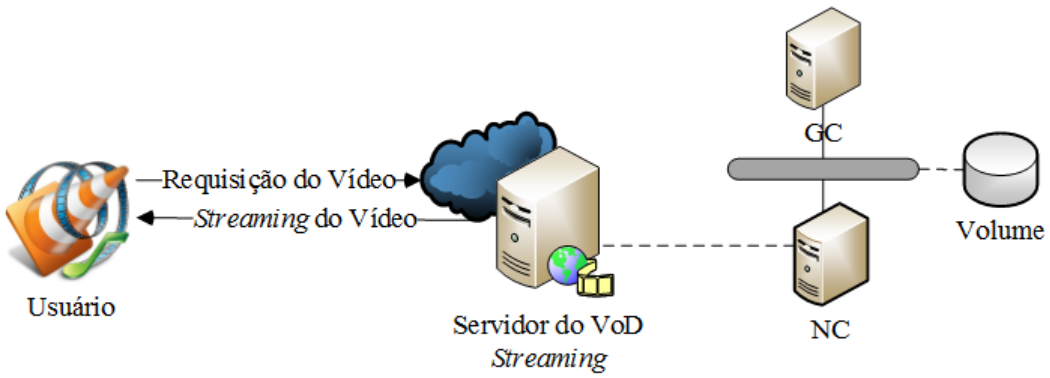


Figura 4.2: Arquitetura Básica

A Figura 4.3 representa o modelo alto nível RBD para a arquitetura básica descrita na Seção 3.2. Para que a infraestrutura dessa arquitetura seja representada, utilizamos composição em série para cada um dos subsistemas do serviço: GC, NC, Volume e Serviço. O GC e o NC são elementos da infraestrutura da plataforma *Eucalyptus*, cujos subsistemas estão representados pelas Figuras 4.4 e 4.5, respectivamente. O Volume representa o dispositivo de armazenamento dos vídeos. O componente Serviço é analisado através de um CTMC específico, apresentado mais adiante na Figura 4.6. Todos estes subsistemas deverão funcionar para que o sistema esteja disponível. Os blocos do modelo que apresentam sombreamento indicam que aquele componente é representado por outro submodelo de forma hierárquica ou por estados, e são descritos após a figura do modelo de alto nível.

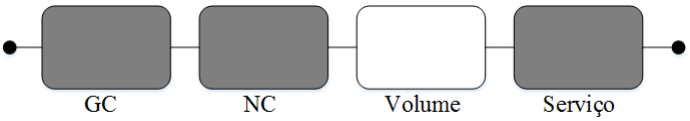


Figura 4.3: Arquitetura básica do serviço - RBD.

O modo de falha do componente GC, representado pela Figura 4.4, é dado pela falha de algum dos seguintes componentes: *Hardware (HW)*, Sistema Operacional (SO), CLC, CC, SC e *Walrus*. Para que o componente GC esteja efetivamente funcionando, é necessário que os componentes de seu subsistema estejam operantes.

Figura 4.4: Subsistema do GC - RBD.

Na sequência, a Figura 4.5 representa o modelo RBD referente ao subsistema de um NC (DANTAS et al., 2012) (BEZERRA et al., 2014). De forma similar ao GC, o subsistema do NC é formado pelos componentes HW e SO, incluindo também o *Kernel-based Virtual Machine (KVM)* e NC:

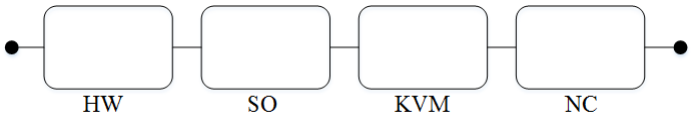


Figura 4.5: Subsistema do NC - RBD.

O bloco Serviço não pôde ser representado através de um modelo RBD por causa das dependências existentes entre os estados dos componentes (DANTAS et al., 2012) pois, o RBD pode não representar a interação entre os estados de um modelo da mesma forma que a modelagem através de CMTC consegue representar. Este formalismo foi utilizado por poder representar bem sistemas de menor complexidade, como é o caso do sistema proposto. Outros formalismos, como o SPN, por exemplo, seriam mais adequados diante de maior complexidade para representar o sistema. Por isso, o componente Serviço é representado através do CTMC da Figura 4.6, que representa a interação entre os componentes *Apache*, *VLC* e *VM*:

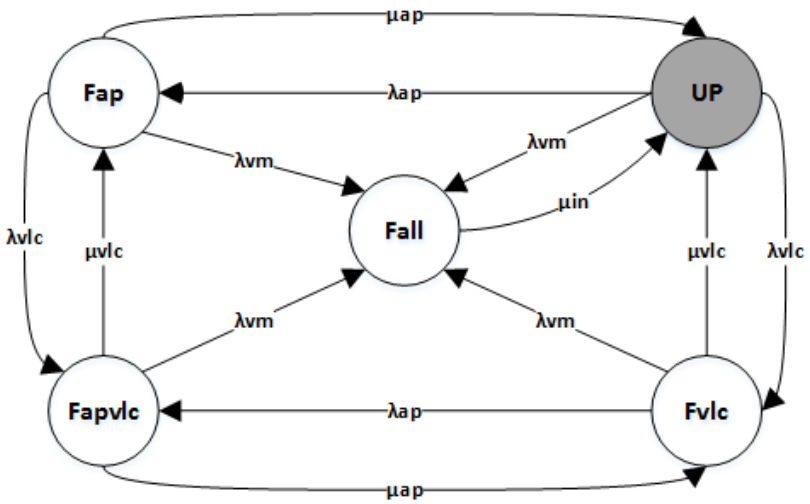


Figura 4.6: Serviço de *VoD streaming* - CTMC.

Essa CTMC representa os cinco prováveis estados que o serviço pode alcançar, são eles: Up, Fap, Fapvlc, Fvlc e Fall. Dentre esses estados, apenas o *Up* indica que todos os componentes envolvidos estão funcionando adequadamente, enquanto que os demais estados representam

falhas no serviço. Quando algum componente alcança um estado de falha, o serviço de *VoD streaming* deixa de ser provido, portanto o sistema alcança um estado de indisponibilidade. A Tabela 4.1 indica o significado de cada sigla dos estados citados:

Tabela 4.1: Estados do CTMC do Serviço.

Estado	Significado
Up	Todos os componentes estão operantes
Fap	Falha na aplicação *Apache*
Fapvlc	Falha na aplicação *VLC*
Fall	Falha em todos os componentes

Para explicar o CTMC que representa o funcionamento do Serviço, vamos começar pelo estado Up, que indica que os componentes estão funcionando e que, por esse motivo, o serviço está disponível. A partir do estado UP é possível que a aplicação VLC falhe, fazendo com o que o estado Fvlc seja alcançado; é possível que a aplicação *Apache* falhe, levando o sistema ao estado Fap; é possível qua a VM falhe, alcançando o estado Fall. Visto que as aplicações Apache e VLC funcionam na VM, a falha dessas aplicações acontece também quando a VM falhar. A partir do estado Fap, onde o serviço não está mais sendo provido devido à falha do *Apache*, o estado Fapvlc pode ser alcançado através da falha da aplicação VLC. Além disso, é possível que haja o reparo da aplicação *Apache*, tornando o sistema disponível no estado UP. É possível também que o estado Fall seja alcançado através da falha da VM. A partir do estado Fapvlc, é possível que haja a recuperação da aplicação *Apache*, fazendo com o que o estado Fvlc seja alcançado; é possível que a recuperação da aplicação VLC, fazendo com o que o estado Fap seja alcançado; é possível que a VM falhe, fazendo com o que o estado Fall seja alcançado. Através do estado Fvlc, é possível que a aplicação VLC seja reparada, tornando o sistema disponível através do estado UP; é possível que haja a falha na VM, alcançando assim o estado Fall; é possível também que a aplicação *Apache* falhe, alcançando o estado Fapvlc. Partindo do estado Fall, que indica que todos os componentes do sistema estão falhos, é possível que haja a instanciação de uma nova VM, tornando o sistema disponível através do estado UP.

Como visto na Figura 4.6, existem três possíveis causas de interrupção do serviço de *VoD streaming*: falha no *Apache*, falha no VLC ou falha na VM. Quando no estado de falha, é necessário que haja o reparo do componente que provoca a indisponibilidade do serviço. No caso do *Apache* e do *VLC*, o tempo de reparo dessas aplicações é superior ao tempo de instanciação de uma nova VM, portanto, quando uma dessas aplicações falha, automaticamente o serviço mata a VM que contém a falha e instancia uma nova VM. A seguir, a Tabela 4.2 apresenta as taxas de falha e reparo referentes ao modelo CTMC para a arquitetura básica do serviço de *VoD streaming*:

Os modelos analíticos alcançados são utilizados para análise de disponibilidade da arquitetura básica. Primeiramente, a partir do CTMC apresentado acima, é possível alcançar a expressão da disponibilidade (Equação 4.2) para calcular a disponibilidade especificamente

Tabela 4.2: Taxas de falha e reparo do serviço - CTMC.

Taxa	Significado
λ_{ap}	Falha do *Apache*
λ_{vlc}	Falha do *VLC*
λ_{vm}	Falha da *VM*
μ_{ap}	Reparo do *Apache*
μ_{vlc}	Reparo do *VLC*
μ_{in}	Instanciação de uma *VM*

do bloco Serviço da arquitetura básica. A expressão da disponibilidade também é necessária para calcular as derivadas parciais necessárias para computar as sensibilidades do sistema. A obtenção dessa expressão foi dada através da ferramenta Mathematica (MATHEMATICA, 2015), cujo processo para concepção da expressão está disponível em (MODCS, 2015).

$$A_s = \frac{(\mu_{in}(\lambda_{ap}\lambda_{vm}(\beta) + \lambda_{ap}(\beta_1)\mu_{vlc} + (\beta_1)(\beta_2)(\beta + \mu_{vlc})))}{((\lambda_{ap} + \beta_1)(\lambda_{vm} + \mu_{in})(\beta)(\lambda_{ap} + \beta + \mu_{vlc}))}, \tag{4.2}$$

Onde:

$$\begin{aligned}\beta &= \lambda_{vlc} + \lambda_{vm} + \mu_{ap},\\ \beta_1 &= \lambda_{vm} + \mu_{ap} \text{ and}\\ \beta_2 &= \lambda_{vm} + \mu_{vlc}.\end{aligned}$$

Para que a disponibilidade de toda a infraestrutura do serviço seja calculada, é necessário utilizar a Equação 4.3, que engloba todos os componentes envolvidos na arquitetura básica. A variável A_S representa a disponibilidade do serviço, e seu valor é obtido a partir da resolução da Equação 4.2 acima. Os valores de A_{GC}, A_{NC} e A_{VOL} são computados a partir de seus valores de MTTF e MTTR aplicados à Equação 2.2. A multiplicação da disponibilidade de cada uma dessas variáveis, resulta na disponibilidade total da arquitetura básica, que é representada por A.

,

$$A = A_{GC} \times A_{NC} \times A_{VOL} \times A_S \tag{4.3}$$

4.2.1.1 Análise de Sensibilidade Paramétrica para a Arquitetura Básica

A análise de sensibilidade é uma técnica importante para avaliação dos efeitos dos parâmetros de entrada nas métricas de interesse, e tem como objetivo a identificação de gargalos de disponibilidade e desempenho do sistema (MATOS JUNIOR et al., 2011; BLAKE; REIBMAN; TRIVEDI, 1988). Visando alcançar melhores resultados de disponibilidade para o serviço, utilizamos o CTMC da Figura 4.6 para calcular os índices de sensibilidade do serviço da arquitetura

básica através da Equação 4.4, que utiliza a derivada parcial e os valores de disponibilidade de cada componente:

$$\begin{aligned} S\theta(A) = & \frac{\partial A_{GC}}{\partial \theta} \times A_{NC} \times A_V \times A_S +, \\ & A_{GC} \times \frac{\partial A_{NC}}{\partial \theta} \times A_V \times A_S + \\ & A_{GC} \times A_{NC} \times \frac{\partial A_V}{\partial \theta} \times A_S + \\ & A_{GC} \times A_{NC} \times A_V \times \frac{\partial A_S,}{\partial \theta} \end{aligned} \tag{4.4}$$

Onde:
A_{GC} significa a disponibilidade do GC;
A_{NC} significa a disponibilidade do NC;
A_V significa a disponibilidade do Volume;
A_S significa a disponibilidade do Serviço.

Os valores das derivadas parciais da equação acima são obtidos a partir dos parâmetros de entrada (taxas de falha e reparo) de cada componente. Para alcançar a derivada parcial do GC, NC, V ou S é necessário aplicar os valores λ e μ de cada componente (um por vez) nas Equações 4.5 e 4.6, substituindo o x pelo componente de interesse:

$$\frac{\partial A\mathrm{f}}{\partial \lambda_f} = -\frac{\mu_x}{(\mu_x + \lambda_x)^2} \tag{4.5}$$

$$\frac{\partial A\mathrm{f}}{\partial \mu_f} = -\frac{\mu_x}{(\lambda_x + \mu_x)^2} + \frac{1}{\lambda_x + \mu_x} \tag{4.6}$$

Os parâmetros de entrada de cada componente são representados pela Tabela 4.3:

Tabela 4.3: Parâmetros de entrada para o ranking de sensibilidade.

Parâmetros	Descrição	Valores (h^{-1})
λ_{NC}	Taxa de falha do NC	$1/481,83$
λ_{GC}	Taxa de falha do GC	$1/180,72$
μ_{GC}	Taxa de reparo do GC	$1/0,9699$
λ_{VLC}	Taxa de falha do VLC	$1/336$
μ_{VLC}	Taxa de reparo do VLC	$1/1$
μ_{NC}	Taxa de reparo do NC	$1/0,910$
λ_{AP}	Taxa de falha do Apache	$1/788,4$
μ_{AP}	Taxa de reparo do Apache	$1/1/1$
μ_{VOL}	Taxa de reparo do Volume	$1/1$
λ_{VOL}	Taxa de falha do Volume	$1/100000$
μ_{IN}	Taxa de instanciar uma nova VM	$1/0,019166$
λ_{VM}	Taxa de falha da VM	$1/2880$

Aplicando os parâmetros de entrada acima nas Equações 4.5 e 4.6 para cada um dos componentes de interesse, podemos alcançar os índices de sensibilidade, cujos valores estão listados na Tabela 4.4, por ordem de importância. O primeiro elemento da tabela é o λ_{NC}, que representa a taxa de falha do componente NC, indicando que melhorias no resultado de disponibilidade total do serviço podem ser alcançadas a partir de alterações nos parâmetros desse componente.

Tabela 4.4: *Ranking* das sensibilidades dos índices da arquitetura básica.

Parâmetro	SS(D)
λ_{NC}	$-1,8850725 \times 10^{-3}$
λ_{GC}	$-1,5866281 \times 10^{-5}$
μ_{GC}	$1,5866281 \times 10^{-5}$
λ_{VLC}	$-8,8157368 \times 10^{-6}$
μ_{VLC}	$8,8126759 \times 10^{-6}$
μ_{NC}	$5,5918029 \times 10^{-6}$
λ_{AP}	$-3,7634921 \times 10^{-6}$
μ_{AP}	$3,7621848 \times 10^{-6}$
μ_{VOL}	$2,9719022 \times 10^{-8}$
λ_{VOL}	$2,9719022 \times 10^{-8}$
μ_{IN}	$1,9777666 \times 10^{-8}$
λ_{vm}	$1,5409453 \times 10^{-8}$

Para exemplificar o passo-a-passo do alcance do resultado da análise de sensibilidade paramétrica utilizada, vamos descrever o alcance da sensibilidade do componente GC. O **primeiro**

passo é calcular a disponibilidade deste componente através dos tempos do *MTTF* e *MTTR* que são, respectivamente: 180,72h e 0,9699h. Esse cálculo é realizado através da ferramenta Mercury (MERCURY, 2014). O resultado da disponibilidade para o bloco GC resulta em 0,994661236. O **segundo passo** desse processo, é converter o MTTF e MTTR deste componente em taxas, onde a taxa de falha do GC (λ_{GC}) será de 0,005533422 (1/MTTF = 1/180,72); e a taxa de reparo do GC (μ_{GC}) será de 1,030938463 (1/MTTR = 1/0,9699). O **terceiro passo** é aplicar essas taxas nas equações 4.5 - com relação ao λ; e 4.6 - com relação ao μ, para obtermos os valores das derivadas parciais com relação ao λ e ao μ do componente GC. Neste caso, o resultado da derivada parcial com relação ao λ é de: -0,959660658. Já a derivada parcial com relação ao μ é de: 0,005150848. O **quarto passo** consiste em calcular as derivadas parciais (tanto λ quanto μ) do GC com relação à disponibilidade dos outros componentes que estão relacionados a ele: GC, Volume e Serviço, aplicando a Equação 4.4. No caso do λ, o cálculo será: -0,959660658 $\times$ 0,998114 $\times$ 0,99999 $\times$ 0,995764273. O resultado dessa multiplicação será de: -0,002834584. O **quinto passo** é encontrar a razão entre o parâmetro e a disponibilidade do sistema. Para o resultado de λ teremos: -0,002834584 $\times$ $\frac{0,005533}{0,9885713}$. Como resultado, chegamos ao valor de $-1,5866281 \times 10^{-5}$, como mostra o *ranking*.

A disponibilidade desta arquitetura básica para o serviço de *VoD streaming* pode ser aumentada, visto que essa arquitetura não apresenta mecanismos redundantes. Com a identificação do componente NC como tendo o maior índice de sensibilidade da arquitetura básica, variações arquiteturais são investigadas com o intuito de melhorar resultados de disponibilidade, através da adoção de mecanismos redundantes, como apresentam as próximas seções.

4.2.2 Arquitetura Redundante - NC

A partir dos resultados de sensibilidade da arquitetura básica, foi identificado que o componente NC tem maior criticidade ao serviço, sugerindo que mudanças nesse componente influenciam nos resultados de disponibilidade total do serviço de *VoD streaming*. Dessa maneira, uma nova arquitetura é proposta a partir da inclusão de um NC redundante à arquitetura básica do serviço de *VoD streaming*.

A arquitetura com redundância no NC tem sua infraestrutura representada pela Figura 4.7, e é composta por três máquinas: uma para o GC e outras duas para o NC. O funcionamento do serviço nessa arquitetura é idêntico ao da arquitetura básica, e agora conta com dois NCs trabalhando em redundância *warm-stand*.

O modo operacional dessa arquitetura é dada pela expressão lógica 4.7 a seguir ($ServicoRed_{up}$), que indica que os componentes GC, Volume e ao menos um NC devem estar operantes para que o serviço de *VoD streaming* esteja efetivamente disponível.

$$ServicoRed_{up} = GC \wedge (NC \vee NC) \wedge Volume \wedge ServidordoVoD \tag{4.7}$$

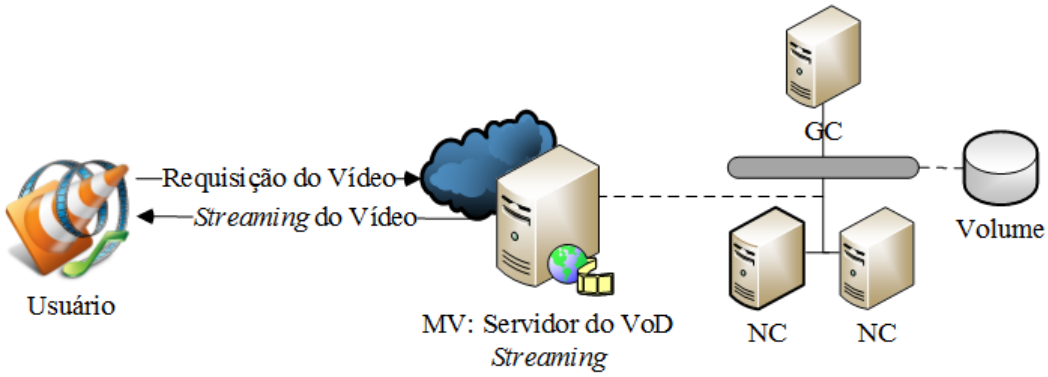


Figura 4.7: Arquitetura Redundante - NC.

A arquitetura com redundância no NC é representada pelo modelo hierárquico RBD da Figura 4.7, e é composta pelos seguintes subsistemas: GC, Volume e Serviço + NC. Todos os subsistemas desse modelo devem estar funcionando de forma correta para que o serviço esteja disponível. Os blocos sombreados indicam que o modelo de falha daquele subsistema é formado por outro submodelo hierárquico.

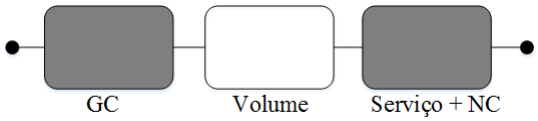


Figura 4.8: Arquitetura com redundância no NC - RBD.

A composição dos subsistemas GC e Volume são as mesmas utilizadas na arquitetura básica apresentada anteriormente. Nesse modelo, o bloco Serviço + NC representa a interação entre o próprio serviço e os NCs redundantes. Portanto, para representar o comportamento desse subsistema, o bloco Serviço é refinado por um modelo baseado em estados (CTMC), representado pela Figura 4.9. Esse CTMC representa a disponibilidade do bloco Serviço em termos de infraestrutura do NC.

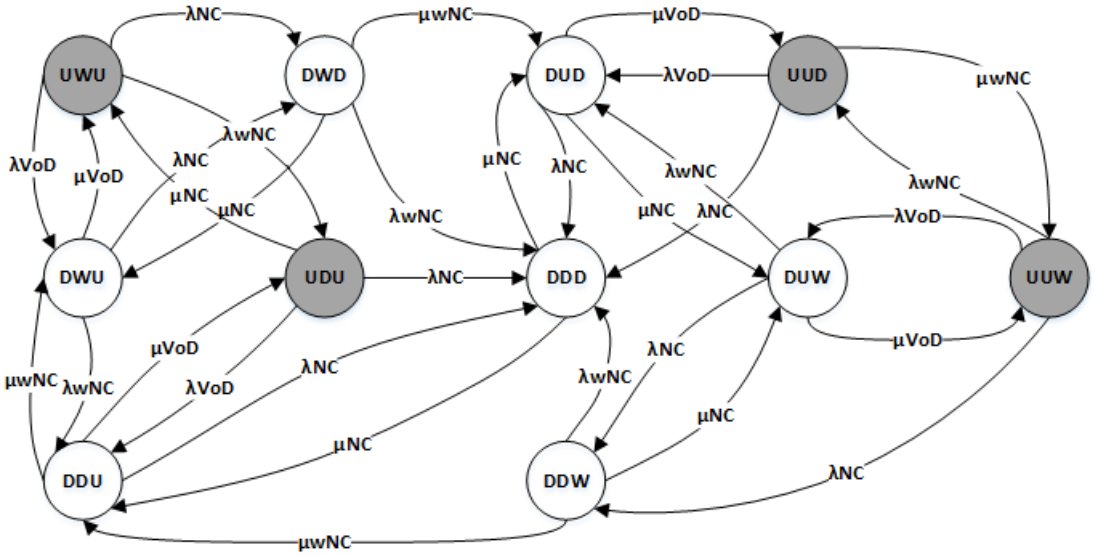


Figura 4.9: Serviço com Redundância no NC - CTMC.

O CTMC acima representa o funcionamento do serviço com uma redundância no NC e possui onze estados possíveis: *UWU*, *UUD*, *UUW*, *UDU*, *DUW*, *DUD*, *DDD*, *DDW*, *DDU*, *DWU* e *DWD*. Essas siglas são formadas por três letras onde cada uma delas indica a condição operacional dos componentes que formam o serviço onde, respectivamente, indicam: o serviço, o primeiro NC e o segundo NC. O serviço pode estar disponível (U) ou indisponível (D). Os NC operam de forma alternada no modo *warm-stand*, e apenas um deles deverá estar *up* (U) por vez, não havendo hierarquia entre eles. Os NCs, ainda, podem assumir a condição de aguardo (W) ou *down* (D). A Tabela 4.5 descreve o significado de cada estado e o *status* do serviço para cada um deles.

Tabela 4.5: Estados do CTMC do Serviço com redundância no NC.

Estado	**Significado**	***Status* do Serviço**
UWU	Serviço disponível, NC em *standby*, NC disponível	Disponível
UUD	Serviço disponível, NC disponível, NC em falha	Disponível
UUW	Serviço disponível, NC disponível, NC em *standby*	Disponível
UDU	Serviço disponível, NC indisponível, NC disponível	Disponível
DUW	Serviço indisponível, NC disponível, NC em *standby*	Indisponível
DUD	Serviço indisponível, NC disponível, NC indisponível	Indisponível
DDD	Serviço indisponível, NC indisponível, NC indisponível	Indisponível
DDW	Serviço indisponível, NC indisponível, NC em *standby*	Indisponível
DDU	Serviço indisponível, NC indisponível, NC disponível	Indisponível
DWU	Serviço indisponível, NC em *standby*, NC disponível	Indisponível
DWD	Serviço indisponível, NC em *standby*, NC indisponível	Indisponível

Nesse modelo, o estado inicial do serviço é representado por UWU e, a partir desse ponto, é possível que haja a falha do NC em *standby*, levando o sistema ao estado UDU; é possível também que haja a falha do NC ativo, levando o sistema ao estado DWD; é possível também haja

a falha no funcionamento do serviço em si, levando o sistema ao estado DWU. Partindo do estado DWD, três outros caminhos são possíveis: falha do NC em *standby*, alcançando o estado DDD; ativação do NC em *standby*, alcançando o estado DUD; reparo do NC em falha, alcançando o estado DWU. A partir do estado DDU, surgem outras três possibilidades: falha do NC ativo, levando o sistema ao estado DDD; reparo do serviço, alcançando o estado UDU; reparo do NC em *standby*, alcançando o estado DWU. Diante do estado DWU, é possível que haja o reparo do serviço, levando o sistema ao estado UWU; que haja a falha do NC ativo, levando o sistema ao estado DWD; que haja a falha do NC em *standby*, alcançando o estado DDU.

No estado DDD todos os componentes do sistema estão indisponíveis (D). A partir desse estado, dois estados podem ser alcançados: reparo do primeiro NC, levando o sistema ao estado DUD, ou o reparo do segundo NC, levando o sistema ao estado DDU. Partindo do estado DUD, outros três caminhos são possíveis: falha do NC ativo, alcançando o estado DDD; reparo e inicialização do serviço, levando ao estado UUD; reparo do segundo NC, levando-o ao modo de *standby* no estado DUW. No estado UUD, é possível que o serviço falhe, levando o sistema ao estado DUD; que que o NC falho seja reparado no modo *standby*, atingindo o estado UUW; ou que haja a falha do primeiro NC, fazendo com o que todos os componentes falhem, como indica o estado DDD.

A partir do estado UDU, é possível que o segundo NC falhe, fazendo com o que todos os componentes estejam falhos, como expressa o estado DDD; que o serviço falhe, alcançando o estado DDU; que o primeiro NC seja reparado e opere no modo *standby*, através do estado UWU. No estado UDU é possível que haja a falha de todos os componentes, alcançando assim o estado DDD; ou o reparo do primeiro NC, assumindo a condição de aguardo, como mostra o estado DWU; ou ainda que uma nova VM seja instanciada tornando o serviço disponível novamente, alcançando o estado UDU.

A partir do estado DUW, é possível alcançar outros três caminhos: falha do NC em standby, alcançando o estado DUD; falha do NC ativo, atingindo o estado DDW, ou ainda o reparo do serviço, como indica o estado UUW. No estado UUW, pode ocorrer a falha do segundo NC levando ao estado UUD, ou ocorrer a falha do serviço alcançando assim o estado DUW ou ainda pode ser que ocorra a falha do primeiro NC, tornando o serviço indisponível no estado DDW. O estado DDW representa o sistema indisponível e, a partir desse estado, é possível que todos os componentes falhem através do estado DDD; que o segundo NC seja inicializado, alcançando o estado DDU; ou ainda que haja o reparo no primeiro NC, caracterizando o estado DUW.

A falha é um evento que ocorre quando o serviço que está sendo provido se desvia do serviço pretendido (AVIZIENIS et al., 2001). As taxas de falha dos dois NCs são representadas por λ_N, enquanto que μ_N representa o reparo desses NCs. O NC em condição de aguardo tem a taxa de falha representada por λ_{wN}. Para que um NC em aguardo assuma a carga de trabalho do serviço, a taxa utilizada é μ_{wN}. A taxa de falha do serviço é λ_{VoD}, enquanto que o reparo do serviço é μ_{VoD}. A ação reparo do serviço é instanciação de uma nova VM, que inclui as

aplicações necessárias ao funcionamento do serviço de *VoD* streaming (*Apache* e *VLC*).

Através dos modelos hierárquicos e analíticos é possível dar início ao processo de alcance dos valores de disponibilidade da arquitetura com redundância no NC. Primeiro, em posse do modelo CTMC da Figura 4.9 acima, podemos obter a Equação 4.8 que representa a exressáo para alcance da disponibilidade do bloco do Serviço com NCs redundantes.

$$A_{SERVICO} = \frac{\alpha(2\beta^2\alpha_1 + \beta\alpha_2\beta_1 + \alpha_3)\beta_1^2 + \alpha_4\beta_1^3}{\alpha_5(\beta^2\alpha_1(\lambda N + 2\mu N) + \beta\varphi\beta_1 + \varphi_1\beta_1^2 + \beta_7\beta^3)}, \tag{4.8}$$

Onde:

$$\begin{aligned}
\beta &= \lambda_{WN},\\
\beta_1 &= \mu_{WN},\\
\alpha &= \mu_N\mu_{VoD},\\
\alpha_1 &= (\lambda_N + \beta + \mu_N)2(\lambda_N + \beta + \mu_N),\\
\alpha_2 &= 4\lambda_{N^2} + 17\lambda_N\beta + 13\beta + 5\lambda\mu_N + 12\beta\mu_N + 2\mu_{N^2},\\
\alpha_3 &= (8\beta(2\beta + \mu_N) + \lambda_N(11\beta + \mu_N)),\\
\alpha_4 &= (7\beta + 2\mu_N)(\mu_{WN^3}),\\
\alpha_5 &= \lambda_N + \lambda_{VoD} + \mu_{VoD},\\
\alpha_6 &= 2\lambda_N(\lambda_N + \beta)(\lambda_N + 3\beta),\\
\beta_2 &= (8\lambda_{N^2} + 24\lambda_N\lambda_{WN} + 13\beta^2)\mu_N,\\
\beta_3 &= (7\lambda_N + 12\beta)\mu_{N^2} + 2\mu_{N^3},\\
\beta_4 &= 2\lambda_N\beta(2\lambda_N + 3\beta),\\
\beta_5 &= (\lambda_N + \beta)(\lambda_N + 16\beta)\mu_N,\\
\beta_6 &= (\lambda_N + 8\beta)\mu_{N^2},\\
\beta_7 &= (\lambda_N(\beta + 2\mu_N) + \mu_N(7\beta + 2\mu_N))\\
\varphi &= \alpha_6 + \beta_2 + \beta_3, and\\
\varphi_1 &= \beta_4 + \beta_5 + \beta_6.
\end{aligned}$$

Para que a disponibilidade de toda a arquitetura com redundância no NC seja obtida, é necessário utilizar a Equação 4.9. Os valores de A_{GC} e A_{VOL} representam a disponibilidade do GC e do Volume, e são computados a partir dos valores de falha e reparo aplicados à Equação 2.2 de disponibilidade. Já $A_{SERVICO}$ é calculado a partir da Equação 4.8 acima. Multiplicando a disponibilidade de cada bloco (GC, VOL e SERVIÇO), alcançamos a disponibilidade total do serviço da arquitetura com redundância no NC, representada por $A_{SERVICOred}$.

$$A_{SERVICOred} = A_{GC} \times A_{VOL} \times A_{SERVICO} \tag{4.9}$$

4.2.2.1 Análise de Sensibilidade Paramétrica para a Arquitetura com Redundância no NC

A partir da Equação 4.8 da expressão da disponibilidade gerada para a arquitetura com redundância no NC, a análise de sensibilidade paramétrica foi aplicada visando identificar o componente mais crítico da arquitetura, a fim de obter melhores resultados de disponibilidade. Os índices de sensibilidade foram calculados a partir da seguinte Equação 4.10, a partir das derivadas parciais e das disponibilidades de cada componente:

$$\begin{aligned} S_\theta(A) = & \frac{\partial A_{GC}}{\partial \theta} \times A_{VOL} \times A_{SERVICO} + , \\ & A_{GC} \times \frac{\partial A_{VOL}}{\partial \theta} \times A_{SERVICO} + \\ & A_{GC} \times A_{VOL} \times \frac{\partial A_{SERVICO}}{\partial \theta} \end{aligned} \tag{4.10}$$

Onde:
A_{GC} significa a disponibilidade do GC;
A_{VOL} significa a disponibilidade do Volume;
$A_{SERVICO}$ significa a disponibilidade do Serviço.

As derivadas parciais foram computadas a partir dos valores de falha e reparo dos componentes, aplicados às Equações 4.5 e 4.6. Os demais valores, como os de falha e reparo, são representados pela Tabela 4.6:

Tabela 4.6: Parâmetros de entrada para a análise de sensibilidade.

Parâmetros	**Descrição**	**Valores** (h^{-1})
λ_{GC}	Taxa de falha do GC	$1/180,72$
μ_{GC}	Taxa de reparo do GC	$1/0,9699$
μ_{VoD}	Taxa de reparo do VoD	$1/0,027$
λ_{NC}	Taxa de reparo do NC	$1/481,83$
λ_{VoD}	Taxa de reparo do VoD	$1/217,779$
μ_{wNC}	Taxa de ativação do NC redundante	$1/0,033$
μ_{VOL}	Taxa de reparo do Volume	$1/1$
λ_{VOL}	Taxa de falha do Volume	$1/100000$
μ_{NC}	Taxa de reparo do NC	$1/0,910$
λ_{wNC}	Taxa de falha do NC redundante	$1/578,196$

Com os índices alcançados, a Tabela 4.7 elenca por ordem de importância o resultado das sensibilidades do serviço com redundância no NC. Observa-se, a partir do resultado, que a taxa μ_{GC} (reparo do GC) aparece no topo do *ranking*, indicando que o componente GC possui a maior criticidade no serviço da arquitetura com redundância no NC.

Tabela 4.7: *Ranking* das sensibilidades dos índices do serviço com redundância no NC.

Parâmetro	SS (A)
μ_{GC}	0,005348791537
λ_{GC}	0,005348791537
μ_{VoD}	0,000184041927
λ_{N}	0,000128199816
λ_{VoD}	0,000126991728
μ_{wNC}	0,000065313366
μ_{VOL}	−0,00001001878
λ_{VOL}	0,000010018748
μ_{NC}	−0,00000748384
λ_{wNC}	0,000001588841

Baseado nesses resultados, uma segunda arquitetura redundante é proposta, dessa vez com uma infraestrutura do serviço de VoD streaming formada por dois GCs e dois NCs redundantes, visando obter melhores resultados de disponibilidade

4.2.3 Arquitetura Redundante: NCs e GCs

A partir da análise de sensibilidade aplicada à arquitetura com redundância no NC, foi identificado que alterações no componente GC influenciam nos valores de disponibilidade do serviço. Dessa forma, essa seção apresenta uma nova arquitetura, tendo a sua infraestrutura composta por um par de NCs redundantes e um par de GCs redundantes, como expressa a Figura 4.10, funcionando no modo *warm-stand*.

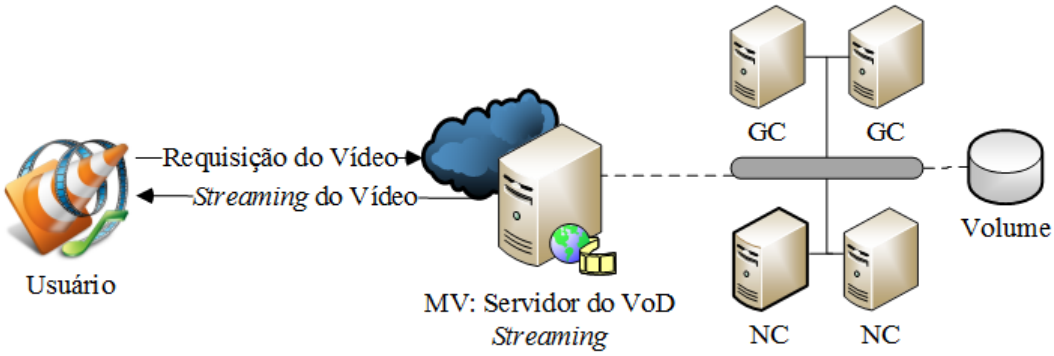


Figura 4.10: Arquitetura com redundâncias no GC e no NC.

O modelo hierárquico RBD que representa a arquitetura com redundância no NC e no GC é formado por três componentes: GC, Volume e Serviço, e também pode ser representado pela Figura 4.8, variando apenas os parâmetros utilizados para tratar o componente GC. O modo operacional dessa arquitetura é expresso pela expressão lógica $ServicoRed_{up}$ 4.11, indicando que ao menos um GC e ao menos um NC, o Volume e o Servidor do *VoD* devem estar operando de

forma adequada para que o serviço esteja disponível. Na Figura 4.11, o CTMC proposto por (DANTAS et al., 2012) representa o comportamento do bloco GC redundante.

$$ServicoRed_{up} = (GC \vee GC) \wedge (NC \vee NC) \wedge Volume \wedge Servidor \tag{4.11}$$

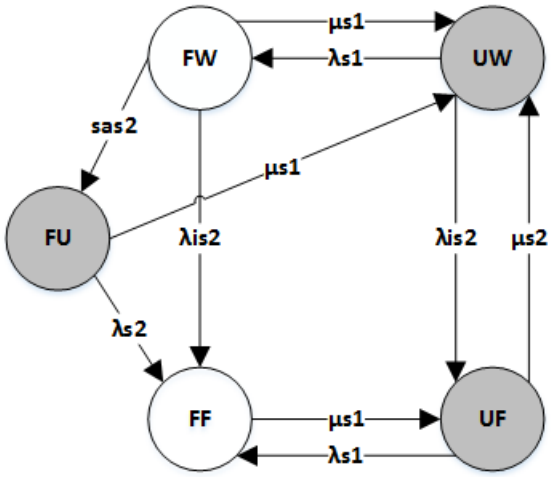


Figura 4.11: Redundância no GC - CTMC.

Existem cinco estados nesse modelo CTMC: UW, UF, FF, FU e FW. Essas siglas representam as iniciais, respectivamente, do estado operacional dos dois GCs, que podem estar ativos (U), no modo de aguardo (W) ou falho (F). Chamamos os componentes de GC1 e GC2, mas não há ordem de prioridade entre os eles. Além disso, apenas um deve estar ativo por vez enquanto o outro está em aguardo ou em falha, garantindo que a função do GC seja mantida no serviço de *VoD streaming*.

Tabela 4.8: Estados do CTMC do Serviço .

Estado	Significado	*Status* do Serviço
UW	GC1 disponível, GC2 em *standby*	Disponível
UF	GC1 disponível, GC2 em falha	Disponível
FF	GC1 em falha, GC2 em falha	Indisponível
FU	GC1 em falha, GC2 disponível	Disponível
FW	GC1 em falha, GC2 em *standby*	Indisponível

Quando o GC1 falha, o sistema alcança o estado FW, onde GC2 ainda não detectou que houve falha no GC1. Se GC2 também falhar nesse intervalo de detecção, então o sistema alcança o estado FF, indicando que GC1 e GC2 estão falhos. O estado FU indica que GC2 não está mais em aguardo e está como ativo no sistema, enquanto GC1 está falho. A partir desse estado, caso GC2 falhe antes que GC1 seja reparado, o sistema alcança o estado FF. Com os dois GCs falhos, o reparo do sistema é acionado, alcançando o estado UW. Através desse estado, é possível que GC2 falhe enquanto GC1 está ativo, onde o sistema alcança o estado UF. A partir desse estado,

podemos observar o reparo do GC2 com o estado UW, ou ainda a falha do GC1, onde o sistema atinge o estado FF.

As taxas de falha referentes aos componentes GC1 e GC2, são representadas por λ_{s1} e λ_{s2}, respectivamente. A taxa $\lambda_i s2$ representa a taxa de falha do GC2 quando ele está inativo, já a taxa de reparo desse componente é μ_{s2}. A taxa de transição sa_{s2} indica a taxa de passagem, ou seja, o inverso do tempo médio para ativar o GC2 após a falha de GC1.

Visando obter a disponibilidade do bloco GC do RBD da arquitetura com redundância no GC e no NC, utilizamos o CTMC da Figura 4.11 para alcançar a Equação 4.12 da expressão da disponibilidade para esse componente.

$$A_{GC} = \frac{\mu(\beta + (\alpha^2) + sa(\lambda + \alpha))}{sa(\lambda^2 + \lambda(\alpha) + \mu(\alpha)) + (\lambda + \mu)(\beta + (\alpha)^2)}, \tag{4.12}$$

Onde:

$$\alpha = \lambda_i + \mu \ ,$$
$$\beta = \lambda \times \lambda_i .$$

Para que a disponibilidade de toda a arquitetura com redundância no NC e GC seja alcançada, é necessário utilizar a Equação 4.13, onde o valor de A_{GC} é obtido a partir da Equação 4.12 acima, A_{VOL} é obtido aplicando-se a fórmula de disponibilidade com os valores de falha e reparo na Equação 2.2 desse componente, e $A_{SERVICO}$ é resultante da Equação 4.8. A multiplicação das disponibilidades de cada bloco, representa a disponibilidade total do serviço de *VoD streaming* com redundância no NC e no GC: A_{rvod}.

$$A_{rvod} = A_{GC} \times A_{VOL} \times A_{SERVICO}, \tag{4.13}$$

4.2.3.1 Análise de Sensibilidade Paramétrica para a Arquitetura com Redundância no NC e no GC

Para que melhores resultados de disponibilidade fossem alcançados, mais uma vez realizamos a análise sensibilidade para compor uma nova proposta de arquitetura para o serviço. Essa análise pôde ser feita a partir da equação 4.12 que representa a expressão da disponibilidade apresentada na Seção anterior. Os índices de sensibilidade foram obtidos através da Equação 4.10, também apresentada na seção anterior. É possível utilizar a mesma equação nas duas situações devido ao fato de as duas arquiteturas apresentarem a mesma composição hierárquica quanto aos seus componentes.

Através das Equações 4.5 e 4.6, aplicamos os valores de falha e reparo dos componentes e obtivemos as derivadas parciais. A Tabela 4.9 apresenta esses parâmetros de entrada:

Tabela 4.9: Parâmetros de entrada para a análise de sensibilidade.

Parâmetros	Descrição	Valores (h^{-1})
μ_{VoD}	Taxa de reparo do Serviço	$1/0,027$
λ_{NC}	Taxa de falha do NC	$1/481,83$
λ_{VoD}	Taxa de falha do Serviço	$1/217,779$
μ_{wNC}	Taxa de ativação do NC redundante	$1/0,033$
μ_{VOL}	Taxa de reparo do Volume	$1/1$
λ_{VOL}	Taxa de falha do Volume	$1/100000$
μ_{NC}	Taxa de reparo do NC	$1/0,910$
λ_{wNC}	Taxa de ativação do NC redundante	$1/578,196$
$\lambda_{s1} = \lambda_{s2}$	Taxa de falha do GC	$1/180,72$
λ_{is2}	Taxa de falha do GC redundante	$1/216,86$
$\mu_{s1} = \mu_{s2}$	Taxa de reparo do GC	$1/0,97$
sa_{s2}	Taxa de ativação do GC redundante	$1/0,005$

A Tabela 4.10 elenca de acordo com a ordem de importância o resultado alcançado nessa análise de sensibilidade. O índice que apresenta maior sensibilidade é o da taxa μ_{s2}, que representa o componente GC. Dessa maneira, o componente GC é o que apresenta maior criticidade à arquitetura com redundância no NC e no GC.

Tabela 4.10: *Ranking* das sensibilidades dos índices do serviço com redundância no GC.

Parâmetro	SS (A)
μ_{s2}	$0,988986686$
λ_{is2}	$-0,982723444$
λ_{VoD}	$-0,350287773$
λ_{s2}	$-0,006229680$
μ_{s1}	$-0,004393499$
λ_{s1}	$0,004365270$
λ_{wN}	$0,002254924$
λ_{N}	$0,001922854$
μ_{N}	$-0,001857340$
μ_{VoD}	$-6,42140 \times 10^{-5}$
$sas2$	$5,35247 \times 10^{-6}$
λ_{VOL}	$-9,99990 \times 10^{-6}$
μ_{VOL}	$9,99990 \times 10^{-6}$
μ_{wN}	$1,06669 \times 10^{-7}$

Com base no que foi identificado com essa análise de sensibilidade, a próxima seção apresenta uma variação na infraestrutura do serviço de *VoD streaming*, onde mais um GC é acrescentado.

4.2.4 Arquitetura Redundante: NCs e GCs - 2

Visando alcançar melhores resultados na nossa análise de disponibilidade do serviço proposto, mais uma vez realizamos a análise de sensibilidade na última arquitetura obtida (Subeção 4.2.3). Diante do resultado alcançado, observamos que mudanças a partir do GC possibilitam ganhos de disponibilidade no serviço. Desta maneira, a Figura 4.12 apresenta uma proposta de arquitetura redundante cuja composição é dada por dois pares de GCs (GC1 e GC2) em redundância *warm-stand* e dois pares de NCs (NC1 e NC2). GC2 e NC2 não funcionam da mesma forma como GC1 e NC1 nessa arquitetura, cuja dedicação é exclusiva ao serviço de *VoD streaming*. Nesse cenário, NC2 e GC2 são recursos que podem ser alocados para outros serviços (*IaaS* ou *SaaS*) enquanto a infraestrutura primária (GC1 e NC1) estiver suportando a carga de trabalho demandada. A partir do momento em que surge uma maior necessidade de recursos para o provimento do serviço de *VoD streaming*, o NC2 é alocado de forma prioritária à infraestrutura principal. Da mesma forma, quando a necessidade de recursos é reduzida à capacidade de um par de NC, o outro par de NC é alocado para atender demandas de terceiros, evitando ociosidade de recursos. Além disso, caso haja a falha dos dois NCs primários, o par de NCs da arquitetura secundária (que também contém o serviço) será acionado. Para que esse gerenciamento pudesse ser realizado de forma mais eficiente, foi necessário adicionar o par de GCs (GC2) à infraestrutura. O funcionamento dos GC2 também acontece sob demanda.

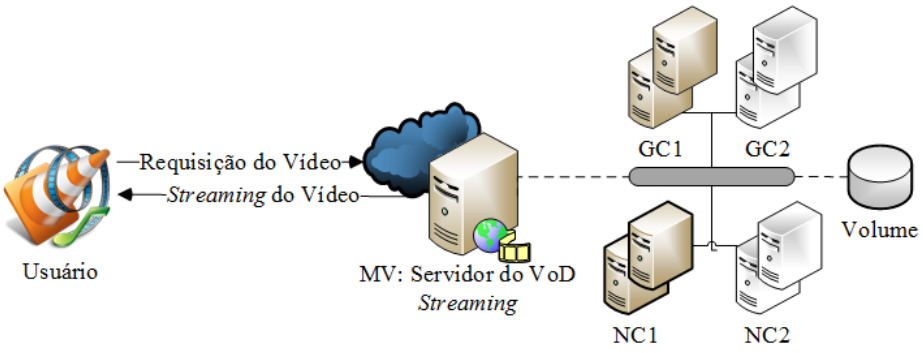


Figura 4.12: Arquitetura redundante com infraestrutura secundária - NC e GC.

O modo operacional que representa a disponibilidade do serviço de *VoD streaming* nessa arquitetura é dada pela Expressão Lógica 4.14, que indica que ao menos um GC, ao menos um NC, o Volume e o Serviço devem estar operando de forma apropriada para que o serviço redundante esteja disponível.

$$ServicoRed_{up} = (GC1 \vee GC2) \wedge ((Servico1 + NC) \vee (Servico2 + NC)) \wedge Volume \qquad (4.14)$$

A representação hierárquica desta arquitetura acontece através do RBD da Figura 4.13, cuja composição é dada por cinco componentes: GC1, GC2, Serviço 1+ NC, Serviço 2 + NC e

Volume. GC1 e GC2 são componentes idênticos ao componente GC apresentado na Subseção anterior (4.2.3), cuja redundância é representada pelo CTMC da Figura 4.11. O componente Serviço 1 + NC também possui o mesmo funcionamento descrito anteriormente (Figura 4.9, assim como o bloco Volume. O bloco Serviço 2 + NC representa dois componentes NCs funcionando de forma redundante, no modo *warm-stand*, e também contém o serviço inserido. A representação dessa redundância é a mesma apresentada na Figura 4.9 anterior.

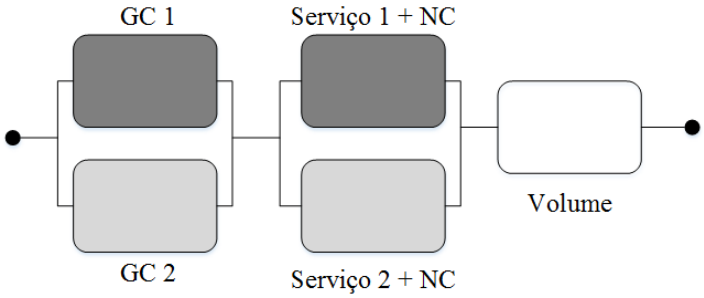


Figura 4.13: Arquitetura com redundâncias no NC e GC - RBD.

4.3 Validação do Modelo

Para verificar se o comportamento do modelo é condizente com o comportamento do ambiente real, realizamos um experimento através de injeção de falhas em um ambiente de testes, a partir da combinação do uso de valores de modelo e do ambiente real. Em seguida, aplicamos cálculos estatísticos propostos por (KEESEE, 1965) para que intervalo de confiança da disponibilidade dos modelos fosse alcançado.

Primeiro, realizamos um experimento de injeção de falhas proposto por (EJLALI et al., 2003), a fim de provocar falhas e reparos nos componentes do sistema. As estratégias de injeção de falhas são utilizadas para validar medidas de disponibilidade do sistema, e englobam a inserção intencional de falhas no sistema, com o objetivo de estudar seus comportamentos de resposta (EJLALI; GHASSEM MIREMADI, 2004). Para injetar as falhas no sistema, elaboramos *scripts* a partir do *Shell Script*, que estão ao final deste trabalho nos Apêndices B, C e D, que representam, respectivamente, os scripts de monitoramento da falha e reparo do *Apache*, *VLC* e *VM*. Esses scripts funcionam de forma bem similar: verificam se o componente está disponível e, caso esteja, lhe aplica um valor exponencial de falha, tendo por base o *MTTF* de cada componente. Depois que o componente estiver falho, o tempo de reparo (*MTTR*) com fator único de redução de 1000 é aplicado para recuperar os componentes envolvidos. A seguir, a Tabela 4.11 apresenta os valores atualizados a partir do fator de redução

Em seguida realizamos a validação dos modelos, para demonstrar que o comportamento dos modelos indica, de forma razoável, o comportamento do sistema real. Utilizamos o método proposto por (KEESEE, 1965), que consiste em calcular o intervalo de confiança da disponibilidade. Aplicamos o método da função de distribuição-F através da ferramenta *Minitab*, que

Tabela 4.11: Fator de redução do MTTF.

Componente	Falha Real (h)	Falha Após Aplicar o F. de Redução (h)	Reparo (h)
Apache	788,4	0,788	0,5
VLC	336	0,336	0,5
VM	2880	2,880	0,019166

considera o número de ocorrências de falhas e reparos no sistema e um intervalo de confiança. O número das ocorrências é explicado na Tabela 4.12, cuja soma resulta no grau de liberdade da validação. O intervalo de confiança que consideramos é de 95%.

Tabela 4.12: Ocorrências de falhas e reparos.

Descrição	Quantidade
Falhas no VLC	135
Reparos no VLC	135
Falhas no Apache	82
Reparos no Apache	82
Falhas na VM	45
Reparos na VM	45
Total de ocorrências	524

A partir do grau de liberdade e do intervalo de confiança de 95% na função de distribuição-F, alcançamos o intervalo de mínimo e máximo da distribuição, como expressa a Tabela 4.13:

Tabela 4.13: Intervalo da Distribuição.

Descrição	Valor
Valor mínimo	0,8425
Valor máximo	1,187

Através dos *scripts* de injeção de falhas, obtivemos os tempos em que o serviço esteve disponível e indisponível no decorrer do experimento. A relação entre o tempo total de falhas e o tempo total de reparos, resulta no valor de ρ, como mostra a Equação 4.15:

$$\rho = \frac{Y_n}{S_n} \tag{4.15}$$

onde Y_n indica o tempo total reparos dos componentes dos serviços, e S_n indica o tempo total de falhas dos componentes dos serviços.

A partir do alcance do valor de ρ, bem como dos valores de mínimo e máximo da distribuição da Tabela 4.13 acima, podemos obter os valores de ρ_L e ρ_U. Esses valores são obtidos a partir da divisão de ρ pelos valores de mínimo e máximo da distribuição, alcançando respectivamente ρ mínino (ρL) e ρ máximo (ρU) (KEESEE, 1965). A Tabela 4.14 os valores de ρ alcançados.

Com os valores de mínimo e máximo de ρ alcançados, podemos obter o intervalo da

disponibilidade. Esse intervalo é calculado a partir das Equações 4.16 e 4.17, que resultam na disponibilidade mínima (A_L) e máxima (A_U) da validação.

$$A_U = \frac{1}{1+\rho U} \tag{4.16}$$

E A_L:

$$A_L = \frac{1}{1+\rho L} \tag{4.17}$$

A Tabela 4.14 reúne os valores de mínimo e máximo de ρ e da disponibilidade:

Tabela 4.14: Intervalo de Confiança para A e ρ.

Intervalo de Confiança (95%)		
ρ	ρ_L	2,3792
	ρ_U	3,3625
A	A_L	0,2292
	A_U	0,2959

Assim, através do cálculo da disponibilidade do modelo analítico que representa o funcionamento do serviço, observamos que a disponibilidade alcançada do serviço foi de 0,2921. Esse valor que se encontra dentro do intervalo da disponibilidade, onde podemos afirmar a validação do modelo, visto que a disponibilidade do sistema real encontra-se dentro dos limites das disponibilidades identificadas no processo de validação.

4.4 Considerações Finais

Este capítulo inicialmente apresentou a metodologia utilizada para elaborar essa pesquisa, através de uma sequência de passos que parte do entendimento do sistema, definição de uma arquitetura básica e uma sequência de variações dessa arquitetura, através da aplicação da técnica de análise de sensibilidade.

Em seguida, foram apresentados as arquiteturas e seus respectivos modelos analíticos. Análise de sensibilidade foi utilizada para identificar gargalos de disponibilidade, sugerindo melhorias para as novas arquiteturas, com base na arquitetura básica. As novas arquiteturas foram modeladas, fornecendo subsídios para a análise de disponibilidade de cada uma delas. Através de CTMCs, o serviço da arquitetura básica e da arquitetura com redundância no NC, bem como o comportamento do GC redundante, foram modelados, tornando possível também o alcance das expressões para cálculo de disponibilidade. Com o intuito de verificar se os modelos representavam o funcionamento do sistema real, foi realizado o processo de validação do modelo.

5

Estudos de Caso

Este capítulo apresenta os estudos de caso que foram elaborados durante o desenvolvimento desta pesquisa. O objetivo destes estudos de caso é realizar a avaliação de disponibilidade dos modelos apresentados na seção anterior, visando alcançar os valores de disponibilidade e *downtime* anual do serviço de *VoD streaming* proposto. Optamos por realizar uma análise de disponibilidade dos modelos analíticos, pelo fato de que métricas como disponibilidade e indisponibilidade anual são muito importantes quando tratamos de provisionamento de algum tipo de serviço. Além dessa análise, realizamos um estudo para observar o comportamento da disponibilidade do serviço diante de variações no tempo de ativação dos componentes redundantes das arquiteturas.

O primeiro estudo de caso realiza a análise de disponibilidade de todas as arquiteturas propostas neste trabalho, onde primeiro tratamos de uma arquitetura básica, sem redundância. Na sequência, observamos o comportamento da disponibilidade do serviço a partir da inclusão de um componente redundante na arquitetura do serviço. Na terceira análise, observa-se os resultados de disponibilidade diante de uma proposta com dois pares de componentes redundantes: GC e NC. Já uma arquitetura com o mesmo tipo de redundância, além de uma infraestrutura secundária, é analisada sob os aspectos de disponibilidade ao final do estudo de caso.

Por fim, visando observar o comportamento da disponibilidade total do serviço diante de instabilidades na rede, o segundo estudo de caso analisa a disponibilidade do serviço a partir de variações no tempo de ativação de componentes redundantes.

5.1 Estudo de Caso I

Este primeiro estudo de caso analisa os resultados de disponibilidade e indisponibilidade anual para as arquiteturas propostas neste trabalho. Inicialmente, vamos abordar a arquitetura básica do serviço de *VoD streaming*, sem redundância. Primeiramente, são apresentados os parâmetros de entrada que foram utilizados para compor cada modelo analítico que representa essa arquitetura. Em seguida, em posse dos modelos analíticos e seus respectivos valores de falha e reparo, pudemos realizar a análise de disponibilidade desse cenário básico com o intuito

de obter as métricas de dependabilidade e indisponibilidade anual. As análises seguintes estão relacionadas às arquiteturas com redunência: NC, NC e GC, e NC e GC + infraestrutura secundária. Para cada uma das arquiteturas redundantes, também são apresentados os valores utilizados para compor os modelos analíticos para cada cenário, seguido da análise de disponibilidade.

5.1.1 Arquitetura básica

A representação da arquitetura básica do serviço de *VoD streaming* é dada pelos modelos analíticos representados pelas Figuras 4.3 e 4.6, apresentados na Seção 4.2.1. Para apresentar os valores de entrada utilizados nos componentes do RBD que representa a arquitetura básica, é necessário primeiro apresentar os parâmetros dos componentes de cada bloco. A Tabela 5.1 concentra as métricas de MTTF e MTTR utilizadas como parâmetros de entrada para composição do modelo RBD do componente GC. Esses valores foram obtidos a partir dos trabalhos (KIM; MACHIDA; TRIVEDI, 2009; HU et al., 2010), e resultaram nos parâmetros de MTTF e MTTR para o GC não-redundante que são, respectivamente, 180,72 horas e 0,96.

Tabela 5.1: Parâmetros de entrada do GC - RBD.

Componentes	MTTF	MTTR
HW	8760 h	150 min
SO	2893 h	15 min
CLC	788,4 h	1 h
CC	788,4 h	1 h
SC	788,4 h	1 h
Walrus	788,4 h	1 h

Em seguida, vamos apresentar os valores que formam os componentes do NC. Uma vez que os componentes GC e NC apresentam as mesmas características de *hardware* e *software*, então Tabela 5.2 apresenta apenas os parâmetros dos componentes KVM e NC (KIM; MACHIDA; TRIVEDI, 2009; HU et al., 2010). Vale ressaltar que os valores de HW e SO para este modelo RBD que representa o NC serão os mesmos considerados na tabela anterior. Como resultado desse modelo RBD, temos um MTTF de 481,82 horas e um MTTR de 0,91 horas, para o bloco do NC.

Tabela 5.2: Parâmetros de entrada do NC - RBD.

Componentes	MTTF	MTTR
KVM	2990 h	1 h
NC	788,4 h	1 h

O bloco Volume não é formado por outros componentes, portanto não é necessária a sua representação por um RBD único. Em posse dos valores de MTTF e MTTR dos componentes de cada bloco do RBD, é possível apresentar a Tabela 5.3, que lista os parâmetros de entrada para o modelo RBD da arquitetura básica. A partir da dessa modelagem, as taxas de falha e

reparo foram obtidas, com as quais o serviço completo de *VoD streaming* pôde ser analisado sob a perspectiva de dependabilidade. Os valores de MTTF e MTTR do Serviço são explicados no próximo parágrafo.

Tabela 5.3: Parâmetros de entrada da arquitetura básica - RBD.

Componentes	MTTF	MTTR
GC	180,72 h	0,969999 h
NC	481,83 h	0,91 h
Volume	100000 h	1 h
Serviço	217,779312 h	0,926338 h

O bloco do Serviço não pôde ser representado através de modelagem hierárquica devido às interações que existem em seus estados. Portanto, foi necessário utilizar modelagem analítica por CTMC para que os valores de MTTF e MTTR fossem alcançados. A Tabela 5.4 indica os parâmetros de entrada que foram utilizados para compor o CTMC da Figura 4.6 que representa o funcionamento do serviço de *VoD streaming* da arquitetura básica, cujos valores foram obtidos em (KIM; MACHIDA; TRIVEDI, 2009; KRSUL et al., 2004):

Tabela 5.4: Parâmetros de entrada do Serviço básico - CTMC.

Parâmetro	Descrição	Taxa (1/h)	Tempo
λ_A	Tempo médio de falha do Apache	0,001268	788,4 h
λ_{VLC}	Tempo médio de falha do VLC	0,002976	336 h
λ_{VM}	Tempo médio de falha da VM	0,000347	2880 h
μ_A	Tempo médio de reparo do Apache	120	0,5 h
μ_{VLC}	Tempo médio de reparo do VLC	120	0,5 h
μ_{in}	Tempo médio de instanciar uma nova VM	0,019166	69 seg

O tempo médio de instanciar uma nova VM, representado pela taxa μ_{in}, foi obtido através de um experimento via *scripts* de coleta de tempos para cada vez que o serviço instanciava uma nova máquina virtual. Através da execução deste *script* observamos que o tempo médio de instaciar uma nova *VM* é de 69 segundos e foi computado através da Equação 5.1. Nesta equação, *MTVMS* significa o *Mean Time to VM Start (MTVMS)* (tempo médio de inicialização da *VM*); e MTSS quer dizer *Mean Time to Service Start (MTSS)* (tempo médio de início do serviço). Ao final deste trabalho, o Apêndice D apresenta o *script* completo para o monitoramento da VM.

$$\mu_{in} = MTVMS + MTSS \tag{5.1}$$

O valor de inicialização da *VM* (*MTVMS*) considera o tempo de instância de uma nova *VM*, e foi alcançado através do seguinte *script*:

```
1    euca-run-instances -t m1.small -z CLUSTER01 emi-5A003F00
2
3    verificaVmUp(){
4    booting=TRUE
```

```
5     while [ $booting = TRUE ]
6     do
7       running=`euca-describe-instances | grep running | wc -l`
8       if [ $running -gt 0 ]
9       then
10        booting=FALSE
11      fi
12    done
13    echo "VM is Running!" >> log-tempos-vm.txt
14    echo "VM is Running!"
15
16     }
17     VerificaVmUp
18     sleep 12
19     ./Mountvol.sh
20     sleep 3
21     ./sstart.sh
22     sleep 5
```

Neste *sccript*, primeiro é solicitada a instanciação de uma *VM small* e, em seguida, a função "VerificaVmUp"busca a VM isntanciada e verifica se ela está disponível, e a verificação é armazenada no arquivo *log-tempos-vm.txt*. O processo de verificação da disponibilidade dura em torno de 10 segundos. Por isso, o *script* Mountvol.sh só será executado após 12 segundos de aguardo. O *script* Mountvol.sh serve para montar o Volume que armazena as mídias do serviço. O *script* de inicialização do serviço, *sstart.sh*, só será executado após 3 segundos. Já o tempo médio de início do serviço considera a preparação do *Volume* do ambiente e a chamada da aplicação *VLC*. O *script* para preparar o Volume é o seguinte:

```
1     /root/credentials/admin/eucarc
2     instancias=`euca-describe-instances | grep INSTANCE | grep running
          | awk '{print $2}'`
3     volumes=`euca-describe-volumes | grep VOLUME | awk '{print $2}'`
4     euca-attach-volume -i $instancias -d /dev/sdf $volumes
5     sleep 10
6     ssh root192.168.10.50 "mount /dev/vdb /vol"
```

Este *script* lista as *VMs* intanciadas e, em seguida, prepara um Volume para a *VM* que está instanciada. O tempo de 10 segundos é o valor médio esperado para que o Volume esteja pronto para ser utilizado. Depois que o Volume está preparado, podemos habilitar a aplicação VLC para o provimento do serviço através do *script*:

```
1 sshpass -p 'cloud123' ssh ubuntu192.168.10.50 "cd /vol ;vlc -ttl 12
      -vvv -color -I telnet -rtsp-host 192.168.10.50 -rtsp-port 8085"
```

Diante dos valores obtidos, foi possível realizar a análise de dependabilidade da arquitetura básica do serviço de *VoD streaming*. Os resultados dessa primeira análise do Estudo de Caso I, em termos de disponibilidade e *downtime* anual, são apresentados na Tabela 5.5.

Tabela 5.5: Resultados de disponibilidade da arquitetura básica.

Medidas	Resultados
Disponibilidade	0,988571
Número de noves	1,9420
Downtime anual	100,12 h

Nesse estudo, apresentamos os parâmetros de entrada para os modelos analíticos que representam o serviço de *VoD streaming* sem redundância, e realizamos a análise de dependabilidade. Como resultado, a disponibilidade encontrada para o serviço na arquitetura básica é de 1,9420 noves, correspondente a uma indisponibilidade anual de 100,12 horas, o que significa 4,17 dias por ano.

5.1.2 Arquitetura com redundância no NC

Nesta segunda análise, primeiramente apresentamos os parâmetros de entrada que foram utilizados para compor os modelos analíticos desse cenário, que contempla a proposta da segunda arquitetura do serviço de *VoD streaming*. Essa arquitetura inclui uma redundância no componente mais crítico da arquitetura básica, o NC, cuja identificação foi possível através do uso da análise de sensibilidade paramétrica. Na sequência, realizamos a análise de dependabilidade dessa arquitetura, visando alcançar melhores resultados de disponibilidade em comparação à arquitetura básica.

Os modelos analíticos que representam o funcionamento dessa arquitetura foram apresentados na Seção 4.2.2. A Tabela 5.6 reúne os parâmetros de falha e reparo utilizados para compor o modelo RBD da Figura 4.8, que representa os componentes da arquitetura com redundância no NC. Os valores do GC e do Volume foram obtidos em (KIM; MACHIDA; TRIVEDI, 2009; HU et al., 2010), e o valor do Serviço é explicado no parágrafo a seguir.

Tabela 5.6: Parâmetros de entrada da arquitetura com redundância no NC - RBD.

Componente	MTTF	MTTR
GC	180,72 h	0,969999 h
Volume	100000 h	1 h
Serviço	149,987435 h	0,037903 h

Mais uma vez foi necessário utilizar um modelo CTMC para representar o comportamento do bloco Serviço, dessa vez com a adição de um componente redundante. Os valores utilizados para compor o CTMC da Figura 4.9, que representa o serviço com redundância no NC, são apresentados na Tabela 5.7. Os valores de λ e μ representam, respectivamente, taxas da falha e reparo. Os valores referentes ao tempo de falha e reparo do NC foram obtidos em (KIM; MACHIDA; TRIVEDI, 2009). O valor de falha referente ao NC em *standby* é o tempo de falha de um NC ativo acrescido de 20% (DANTAS et al., 2012), já que ele encontra-se ligado, mas

não operacional; já o de reparo é o tempo de ativação do componente em *standby*, que acontece em um tempo médio de dois minutos com uso da aplicação *Heartbeat* (HEARTBEAT, 2014). Os tempos de falha do Serviço é resultado do CTMC apresentado no estudo anterior, já o tempo médio de instanciar o Serviço é explicado no próximo parágrafo.

Tabela 5.7: Parâmetros de entrada do serviço com redundância no NC - CTMC.

Parâmetro	Descrição	Taxa (1/h)	Tempo
λ_{N1}=λ_{N2}=λ_N	Tempo médio de falha do NC	0,002075	481,80 h
λ_{wN1}=λ_{wN2}=λ_{wN}	Tempo médio de falha do NC em *standby*	0,001729	578,19 h
μ_{N1}=μ_{N2}=μ_N	Tempo médio de reparo do NC	1,098901	0,91 h
μ_{wN1}=μ_{wN2}=μ_{wN}	Tempo médio de reparo do NC em *standby*	30,3030	0,33 h
λ_{VOD}	Tempo médio de falha do Serviço	0,004591	217,77 h
μ_{VOD}	Tempo médio de instanciar o Serviço	37,037037	0,027 h

O tempo de médio de instanciar o Serviço é representado pela taxa μ_{VOD}, que resulta da Equação 5.2:

$$\mu_{VOD} = TNODES + \mu_{in}, \tag{5.2}$$

onde TNODES é o tempo médio para ativar o NC, cujo valor é obtido através da aplicação *Heartbeat* (HEARTBEAT, 2014). A funcionalidade dessa aplicação consiste em enviar mensagens de um NC para o outro NC, detectando quando o NC ativo der sinais de falha (não respondendo a mensagem), para que o NC em *standby* possa ser ativado, sem que haja interrupção ou atraso crítico para o usuário final. A taxa μ_{in} é extraída da Equação 5.1 que representa o tempo de instanciar uma nova VM, e foi explicada no estudo de caso anterior.

A partir dos valores correspondentes a esse segundo estudo, é possível realizar a análise de dependabilidade alcançando os resultados de disponibilidade, como apresenta a Tabela 5.8.

Tabela 5.8: Resultados de disponibilidade da arquitetura com redundância no NC.

Medidas	Resultados
Disponibilidade	0,994401
Número de noves	2,5118
Downtime anual	49,05 h

Este segundo estudo teve como objetivo observar o comportamento da disponibilidade do serviço proposto diante da inclusão de um componente NC em redundância *warm-stand*. Primeiro, apresentou-se os parâmetros de entrada para os modelos analíticos que representam essa arquitetura redundante. Em seguida realizou-se a análise de dependabilidade e foi observado um aumento na disponibilidade do serviço de *VoD streaming* em comparação ao estudo anterior. Diante dessa proposta com redundância no NC, a disponibilidade total do serviço passa a ser

Tabela 5.9: Parâmetros de entrada do serviço com redundância no NC e no GC - CTMC.

Componente	MTTF	MTTR
GC Redundante	99,953747 h	0,460964 h
Volume	100000 h	1 h
Serviço	149,987435 h	0,037903 h

de 2,5118 noves, o que representa uma diminuição no tempo de *downtime* anual do serviço em 51,01%, em comparação à arquitetura básica.

5.1.3 Arquitetura com redundância no NC e no GC

Essa análise é resultante da análise de sensibilidade paramétrica aplicada à arquitetura com redundância no NC, que identificou que o componente GC tem maior criticidade ao modelo, podendo influenciar na melhoria do resultado de disponibilidade do serviço de *VoD streaming*. O objetivo desse estudo, é apresentar os valores de falha e reparo dos componentes que formam os modelos analíticos dessa proposta e, em seguida, realizar uma análise de dependabilidade com o objetivo obter os resultados de disponibilidade e indisponibilidade anual, e observar a evolução desses resultados em comparação aos resultados dos estudos anteriores.

A Seção 4.2.3 apresentou os modelos analíticos cujos parâmetros serão apresentados aqui. O modelo RBD que representa essa arquitetura redundante é graficamente igual ao que foi apresentado no estudo anterior, variando apenas os parâmetros de falha e reparo do bloco GC, que dessa vez corresponde ao funcionamento de um par de GCs com redundância. Os valores desse bloco serão explicados no parágrafo a seguir. A Tabela 5.1 concentra os parâmetros utilizados para cada bloco do modelo.

A redundância do GC foi proposta por (DANTAS et al., 2012), e os valores que formam esse componente foram obtidos a partir da modelagem de seu comportamento através de CTMC (Figura 4.11, de acordo com os parâmetros indicados na Tabela 5.10. As taxas de tempo médio de falha do GC ativo, sem hierarquias de um para o outro, são representadas por λ_{s1} e λ_{s2}. O tempo médio de reparo do GC, independente se for para ficar ativo ou ficar em modo *standby*, tem o mesmo valor e é atribuído a μ_{s1} e μ_{s2}. Esses valores de falha e reparo foram obtidos através do modelo RBD da Figura 4.4. A falha do GC em *standby* é representada por λ_{is2}, que pode ocorrer, inclusive, no intervalo de tempo de ativação do GC que está em processo de assumir a carga de trabalho do serviço. A taxa sa_{s2} representa o tempo de ativação do serviço e foi extraído da aplicação *Heartbeat* (HEARTBEAT, 2014), cujo valor padrão de ativação do componente redundante é de 0,005 horas, ou aproximadamente 0,18 segundos.

A partir dos valores obtidos nesse estudo de caso, podemos apresentar a Tabela 5.11, que contempla os resultados de dependabilidade obtidos para esta arquitetura.

Nessa análise, apresentamos os valores de falha e reparo utilizados para compor os modelos analíticos que representam a proposta da arquitetura com redundância tanto no GC, quanto no NC. Com esses valores, alcançamos a análise de dependabilidade dessa proposta

Tabela 5.10: Parâmetros de entrada do GC redundante - CTMC.

Parâmetro	Descrição	Taxa (1/h)	Tempo
$\lambda_{s1} = \lambda_{s2}$	Tempo médio de falha do GC ativo	0,005533	180,72 h
λ_{is2}	Tempo médio de falha do GC em *standby*	0,004611	216,86 h
$\mu_{s1} = \mu_{s2}$	Tempo médio de reparo do GC ativo	1,030927	0,97 h
sa_{s2}	Tempo médio para ativar o GC	200	0,005 h

Tabela 5.11: Resultados de disponibilidade da arquitetura com redundância no NC e no GC.

Medidas	Resultados
Disponibilidade	0,995113
Número de noves	2,3109
Downtime anual	42,81 h

redundante, cujo resultado indicou uma disponibilidade de 2,3109 noves, o que representa um tempo de *downtime* anual de 42,81 horas, ou 1,78 dias por ano. Esse resultado apresenta um bom ganho de disponibilidade quando comparado à arquitetura sem redundância: 57,25%. Porém, essa redução na indisponibilidade anual do serviço não é tão expressiva quando comparada à arquiteura com redundância apenas no NC: 12,73%.

5.1.4 Arquitetura com redundância no NC e no GC + infraestrutura secundária

Este estudo compreende uma arquitetura redundante elaborada a partir dos resultados da análise de sensibilidade realizada na Seção 4.2.3.1, indicando que a partir de mudanças no GC, melhores resultados de disponibilidade seriam obtidos. Assim, esse estudo apresenta os parâmetros de falha e reparo que formam os modelos analíticos dessa arquitetura, e realiza a análise de dependabilidade com o intuito de obter os valores de disponibilidade e indisponibilidade anual dessa proposta redundante. Ao final desse estudo, todos os resultados alcançados são analisados e comparados.

Os modelos analíticos que representam essa arquitetura foram apresentados anteriormente, na Seção 4.2.4. A composição do modelo hierárquico RBD da Figura 4.13 é dada pelos parâmetros de falha e reparo de alguns componentes já apresentados na Tabela 5.6. GC1 e GC2 tem os mesmos valores que GC; Volume é o mesmo componente apresentado nos estudos de caso anteriores, cujos valores de MTTF e MTTR estão na Tabela 5.9. O bloco Serviço também é o mesmo componente apresentado anteriormente, cujos valores de falha e reparo foram apresentados anteriormente na Tabela 5.7.

A partir dos valores apresentados, foi possível alcançar a análise dos resultados dessa arquitetura redundante para o provimento do serviço de *VoD streaming*. Os resultados de dependabilidade (disponibilidade e *downtime* anual) são mostrados na Tabela 5.12:

Visando uma análise dos estudos apresentados até o momento, a Figura 5.1 realiza uma

Tabela 5.12: Resultados de disponibilidade da arquitetura com redundância no NC e no GC + infraestrutura secundária.

Medidas	Resultados
Disponibilidade	0,999846
Número de noves	3,8146
Downtime anual	1,3490 h

comparação dos quatro resultados de disponibilidade das arquiteturas propostas, considerando valores de disponibilidade e tempo de *downtime* anual. A arquitetura básica demonstra o maior nível de indisponibilidade, indicando um valor de disponibilidade de 0,988571. Para a arquitetura do NC redundante, o serviço de *VoD streaming* apresenta-se mais disponível, alcançando uma diminuição de 51,01% na indisponibilidade do serviço.

Em seguida, a arquitetura que apresenta uma arquitetura composta por GC e NC redundantes em *warm-stand*, demonstra um aumento no resultado de disponibilidade, com 0,995113 noves, o que representa uma redução de 57,25% da indisponibilidade anual do serviço quando comparado aos resultados da arquitetura básica. Com a adição de uma infraestrutura secundária, a arquitetura com a proposta do GC e NC redundantes além da infraestrutura secundária, apresentou os melhores resultados com relação à disponibilidade do serviço, cujo valor foi de 3,81 noves, o que impacta em uma redução de 98,66% na indisponibilidade do serviço em comparação à primeira arquitetura proposta.

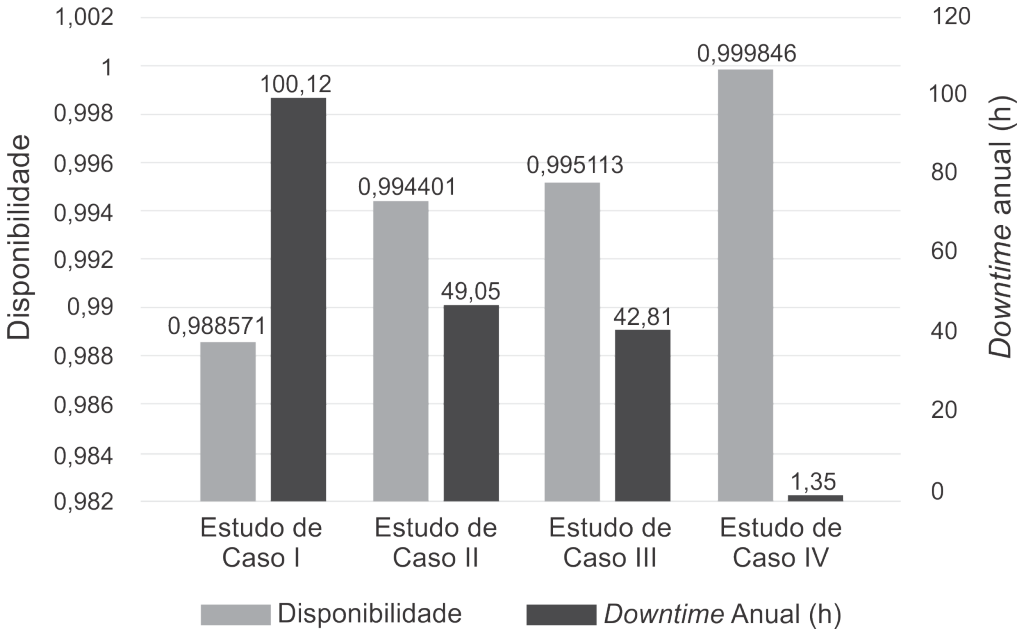


Figura 5.1: Comparação da disponibilidade das arquiteturas propostas.

5.2 Estudo de Caso II

Um componente redundante precisa de um tempo *X* para que assuma a atividade do sistema diante de uma falha no componente principal. Porém, situações internas ou externas podem provocar a variação desse tempo de ativação do componente redundante. Este estudo de caso analisa o impacto na disponibilidade total do serviço quando o tempo de ativação dos componentes redundantes sofrer variações dentro de uma escala de dez minutos. O objetivo é observar qual componente do serviço que traz maior impacto na disponibilidade total diante dessas variações de tempo.

O tempo de ativação de um componente redundante é um fator crítico para sistemas como o de *VoD streaming*, pois existe a possibilidade de que eventos internos ou externos possam comprometer esse tempo de ativação causando, por exemplo, quebras de acordos de nível de serviço (SLAs), dentre outros prejuízos. De acordo com a aplicação *Heartbeat*, o tempo de ativação para um componente redundante é de dois minutos (DANTAS et al., 2012), (HEARTBEAT, 2014). No entanto, diante de situações que possam, por exemplo, aumentar o fluxo na rede causando um comprometimento no tempo de ativação dos componentes redundantes, valores de disponibilidade do serviço de *VoD streaming* podem ser impactados diretamente. Portanto, o tempo de ativação dos componentes redundantes é variado em uma escala de 2 minutos até 12 minutos observando os efeitos que essa variação pode trazer à disponibilidade de todo o sistema.

A Figura 5.2, demonstra os efeitos da variação do tempo de ativação do NC redundante na disponibilidade total do sistema funcionando na arquitetura com redundância no NC. Como esperado, à medida que o tempo de ativação do componente redundante aumenta, a disponibilidade do sistema diminui, já que o serviço ficaria mais tempo em aguardo para ativação, ou seja, o serviço estaria indisponível por mais tempo. A disponibilidade total do serviço alcança 2,2528 noves quando o tempo de ativação do NC redundante é de 2 minutos, e decresce para 2,2311 quando esse tempo atinge 12 minutos, o que significa um *downtime* anual de 49,06 horas (para 2 minutos) até 51,04 horas (para 12 minutos). Dessa forma, podemos afirmar que, a cada minuto de atraso no tempo de ativação do componente NC redundante, representa um acréscimo médio de 0,198 horas na indisponibilidade anual total do serviço de *VoD streaming*.

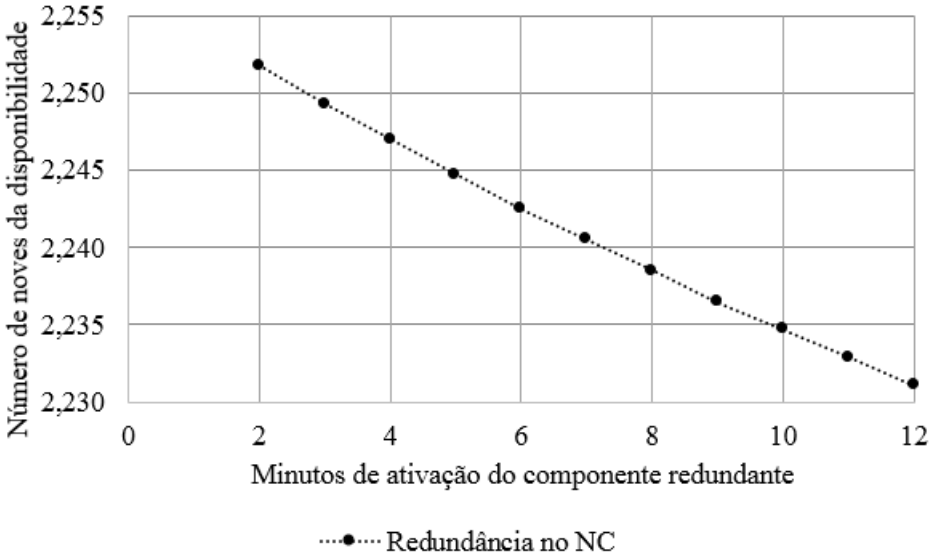


Figura 5.2: Variação no tempo de ativação do NC.

Com relação ao componente redundante GC, a Figura 5.3 representa o comportamento da disponibilidade total do serviço de *VoD streaming* perante a variação no tempo de ativação deste componente. A disponibilidade do serviço de *VoD* streaming, variando apenas o GC redundante, vai de 2,3109 noves (2 minutos) até 2,2512 (12 minutos), o que equivale a um *downtime* anual de 42,80 até 49,12 horas, respectivamente. Isso indica que a variação no tempo de ativação desse componente impacta mais na disponibilidade total do serviço em comparação à análise anterior. Dessa forma, podemos observar uma média de 0,632 horas de acréscimo na indisponibilidade total do serviço a cada minuto de atraso na ativação do GC redundante.

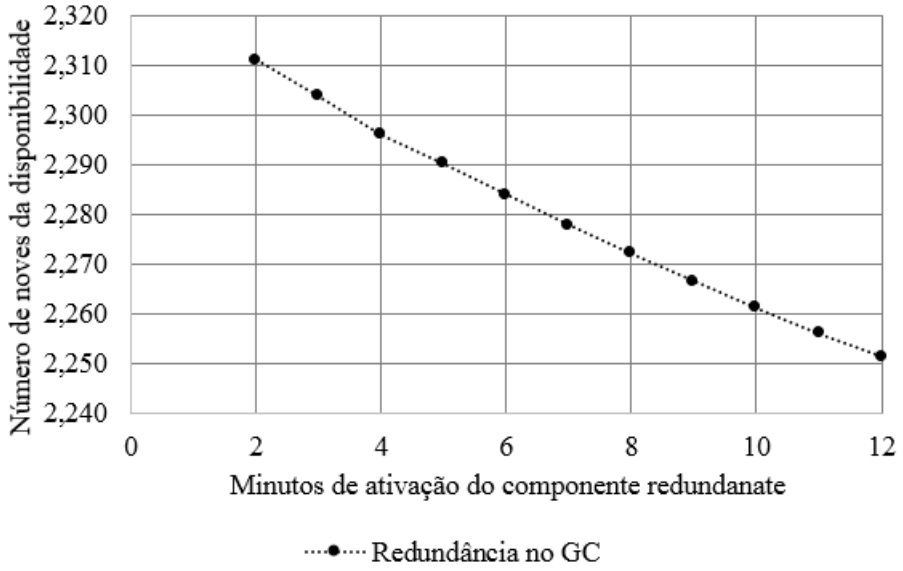


Figura 5.3: Variação no tempo de ativação do GC.

A Figura 5.4 combina os resultados dos dois gráficos anteriores para ilustrar os efeitos da disponibilidade do serviço de *VoD streaming* quando os tempos de ativação dos dois componentes

redundantes (NC e GC) sofrem variação de 2 a 12 minutos. A disponibilidade total desse cenário agora varia de 2,3109 (2 minutos) até 2,2305 (12 minutos) noves, variando o tempo de *downtime* anual de 42,80 até 51,51 horas. Em outras palavras, isso significa dizer que a indisponibilidade anual do serviço pode aumentar em até 20.35% diante de uma variação no tempo de ativação dos componentes redundantes do serviço de até 12 minutos. Vale observar também que a variação no tempo de ativação do NC redundante não afeta tanto a disponibilidade total do serviço quanto a variação no tempo de ativação do GC redundante, sugerindo que quanto maior for o tempo de ativação do GC, maior será a indisponibilidade total da arquitetura que utilizar redundância no GC. Ou seja, é preciso que o tempo de ativação do GC redundante seja o menor possível, para que se observe melhores resultados de disponibilidade total do serviço.

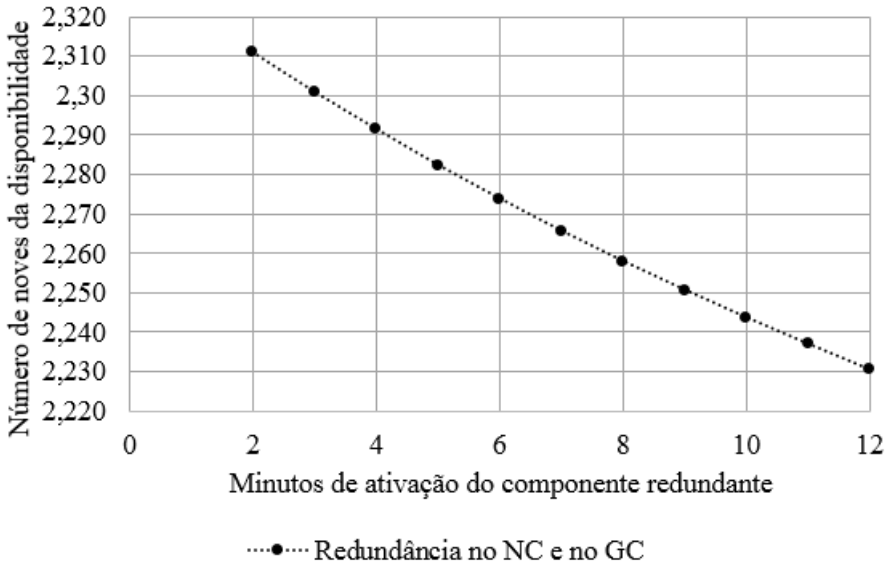


Figura 5.4: Variação no tempo de ativação do NC e do GC.

5.3 Considerações Finais

Esse capítulo trata de dois estudos de caso elaborados para analisar aspectos de disponibilidade em quatro diferentes arquiteturas para provimento do serviço de *VoD streaming*. A partir dos resultados obtidos, é possível ter maior compreensão a respeito dos fatores que influenciam positivamente ou negativamente a disponibilidade de tais arquiteturas.

O primeiro estudo de caso desse capítulo refere-se à análise de disponibilidade das arquiteturas propostas. A primieira análise está relacionada à arquitetura básica do serviço de *VoD streaming*, sem redundâncias, e apresenta os parâmetros utilizados na composição dos modelos RBDs e CTMC, utilizados para o cálculo das métricas de dependabilidade (disponibilidade e indisponibilidade anual). Como resultado, é possível verificar que a arquitetura básica, sem redundâncias, apresenta a menor disponibilidade entre as arquiteturas apresentadas. Em seguida, na segunda análise desse estudo, é apresentada a arquitetura com redundância no NC, sugerida a partir de análise de sensibilidade aplicada à arquitetura do estudo anterior. Ainda na

segunda análise são apresentados os valores de falha e reparo dos componentes que formam a infraestrutura redundante, resultando nos valores de disponibilidade do serviço, que indicam uma melhoria quando comparados aos resultados da arquitetura básica. A terceira análise é realizada a partir do componente de maior criticidade do estudo anterior, sugerindo a adição de um GC redundante à infraestrutura do serviço. Com isso, a terceira análise desse estudo apresenta os valores de falha e reparo dos componentes que formam essa infraestrutura, alcançando os resultados de disponibilidade dessa arquitetura. Assim, é possível observar que a arquitetura com redundância no GC e no NC apresentam os melhores resultados de disponibilidade, chegando a reduzir 57,25% a indisponibilidade anual do serviço quando comparado aos resultados da arquitetura básica. Na sequência, a quarta análise de disponibilidade apresentou as métricas de falha e reparo dos componentes que formam a arquitetura em foco, cuja composição é dada pela redundância dos componentes GC e NC, além de apresentar uma infraestrutura secundária também formada pela redundância dos componentes GC e NC. Como resultado da análise dessa proposta, observou-se uma redução de 99,99% na indisponibilidade do serviço, em comparação à primeira proposta.

O segundo estudo de caso considera que eventos internos ou externos podem variar o tempo de ativação do componente redundante, impactando diretamente a disponibilidade total do serviço. Desta forma, esse estudo de caso analisa uma variação entre dois e doze minutos no tempo de ativação do GC, do NC e nos dois componentes juntos, observando o comportamento da disponibilidade total. Dessa forma, é possível observar que a variação do NC tem menor impacto na disponibilidade, já o GC apresenta maior impacto na disponibilidade total quando o seu tempo varia. Assim, quanto menor o tempo de ativação do GC, maior a disponibilidade total do serviço de *VoD streaming*. Como resultado, é possível observar que as arquiteturas com redundância apresentam melhores resultados de disponibilidade, visto que diante de uma falha de um componente de maior criticidade, um outro componente assume a carga de trabalho, amenizando o tempo de inatividade do serviço.

6

Conclusões e Trabalhos Futuros

O crescimento da adoção da computação em nuvem possibilita que diversos tipos de serviços se beneficiem desse paradigma, através da redução dos custos de implantação e da elasticidade sob demanda, por exemplo. Exemplos mais notáveis desses tipos de serviços são os de armazenamento de arquivos, transações bancárias e *streaming* de músicas ou vídeos. Um grande provedor de *streaming* de vídeo utilizando a nuvem é o *Netflix*, que hoje soma mais de 53 milhões de usuários únicos distribuídos no mundo todo (NETFLIX, 2014b), e esse número só tende a aumentar. Porém, mesmo diante de tanta evolução e tanto investimento nesse modelo computacional que é a nuvem, ainda não é possível garantir que um sistema esteja isento de ocorrências de falhas, uma vez que serviços computacionais demandam alta disponibilidade. Diante disso, a identificação de gargalos de disponibilidade é uma boa técnica para amenizar os impactos de falha no sistema onde, por exemplo, componentes redundantes podem alcançar melhores disponibilidades.

Desta forma, este trabalho realizou um estudo de avaliação de disponibilidade em quatro diferentes arquiteturas para o provimento de um serviço de vídeo *streaming* na nuvem, baseado em uma infraestrutura de nuvem privada, com o objetivo de alcançar melhorias na disponibilidade. O primeiro deles estava relacionado à uma arquitetura básica, sem redundância, cujo obetivo principapal foi apresentar os modelos analíticos que representasse o comportamento dessa arquitetura, bem como alcançar o resultado de disponibilidade total desse serviço, através dos valores de falha e reparo de cada um dos componentes que formam essa proposta sem redundância. Para isso, a representação desse cenário foi realizada através de modelagem hierárquica com CTMC e RBD. Através dos modelos analíticos, a disponibilidade de 0,988571 noves para a arquitetura básica foi alcançada. Para o primeiro cenário, foi aplicada a técnica de análise de sensibilidade paramétrica, cujo o objetivo foi de indentificar gargalos de disponibilidade nessa proposta sem redundância. Através dessa análise, identificou-se que o NC é o gargalo de disponibilidade da arquitetura básica. Assim, o segundo cenário tratou de analisar o resultado de disponibilidade para uma arquitetura com adição de um NC em redundância *warm-stand*. A representação dessa arquitetura foi dada através dos modelos analíticos, cujo resultado de disponibilidade para essa arquitetura redundante apresentou uma redução de 51,01% de *downtime* anual em comparação

ao primeiro cenário. Análise de sensibilidade foi aplicada para essa arquitetura com redundância no NC, indicando que o GC é o componente de maior criticidade para essa proposta.

Assim, o terceiro cenário tratou de uma arquitetura com redundância tanto NC quanto no GC, cujo resultado de disponibilidade foi alcançado através dos modelos analíticos para essa arquitetura. A indisponibilidade anual para essa proposta foi reduzida em 57,25% com relação à arquitetura sem redundância. Mais uma vez a análise de sensibilidade foi aplicada, identificando o GC como o mais crítico da arquitetura. O quarto cenário tratou da análise de disponibilidade de uma arquitetura com dois GCs redundantes, dois NCs redundantes, além de uma infraestrutura secundária. Como resultado, essa arquitetura apresentou uma redução de 99,99% da indisponibilidade anual em comparação à arquitetura básica, sem redundância. Todos esses cenários contemplam o Estudo de Caso I. É importante destacar que o modelo que representa a funcionalidade do serviço foi validado, garantindo assim que os modelos analíticos do serviço proposto apresentam comportamento próximo ao de um sistema real. As avaliações de cada um dos cenários obtidos foram realizadas a partir de modelagem hierárquica baseada em RBD e CTMC.

O segundo estudo de caso analisou o impacto do tempo de ativação dos componentes redundantes na disponibilidade total do serviço em um intervalo entre dois e doze minutos. Foi percebido que quando o tempo de ativação do componente NC redundante aumenta, a disponibilidade total do serviço não sofre tanto impacto em comparação a quando há variações no tempo de ativação do componente GC redundante. Os resultados desse estudo mostraram que, quanto menor o tempo de ativação do componente GC, maior será a disponibilidade total do serviço, por ele ter maior criticidade que o componente NC.

Os resultados obtidos durante esta pesquisa podem auxiliar provedores de serviço de *VoD streaming* na nuvem a tomar decisões com relação ao tipo de arquitetura que poderá ser adotada dependendo da disponibilidade que se deseja. Cada arquitetura possui suas particularidades de capacidade e custos, portanto quanto maior o número de componentes da arquitetura, maior será o custo.

6.1 Contribuições

Como resultado do que foi apresentado nessa dissertação, é possível destacar as seguintes contribuições:

Proposições de modelos analíticos (hierárquicos e baseados em estados) para representar o funcionamento de um serviço de *VoD streaming* baseado na plataforma de nuvem *Eucalyptus*. Através destes modelos, foi possível analisar métricas de dependabilidade deste serviço como disponibilidade e indisponibilidade anual. Tais modelos podem ser utilizados para a representação deste tipo de serviço em propostas futuras.

Aplicação da técnica de análise de sensibilidade paramétrica a partir da arquitetura mais básica do serviço, gerando um *ranking* de sensibilidade dos parâmetros de cada arquitetura

proposta, indicando gargalos de disponibilidade em cada cenário. A partir desse *ranking* de sensibilidades, foi possível guiar a concepção de novas arquiteturas para o serviço de *VoD streaming*, com o objetivo de alcançar melhores resultados de disponibilidade. Além disso, esses resultados das sensibilidades podem ser utilizados para guiar outros trabalhos que analisem infraestruturas do *Eucalyptus*.

Proposta de quatro variações de arquiteturas (uma básica e outras três com redundância) para o provimento do serviço de *VoD streaming* considerando variações na infraestrutura a partir da adoção de mecanismos redundantes. Essas arquiteturas, bem como os modelos analíticos que representam essas arquitetuas, podem ser utilizados em extensões de outras arquiteturas para esse serviço em trabalhos futuros.

A análise dos benefícios da adoção de mecanismos de replicação *warm-stand* em um serviço de *VoD streaming* na nuvem baseado na plataforma *Eucalyptus*, em termos de dependabilidade, considerando as métricas de disponibilidade e indisponibilidade anual. Também destacamos o alcance das expressões para cálculo de disponibilidade dos modelos CTMCs propostos para o serviço de *VoD streaming* com base na plataforma *Eucalyptus* - sem redundância e com redundância.

Além das contribuições mencionadas acima, foram desenvolvidos alguns artigos com base nos resultados dessa pesquisa, como principal autoria e como colaboração:

- Maria Bezerra, Rosangela Melo, Jamilson Dantas, Paulo Maciel, and Francisco Vieira, "*Availability modeling and analysis of a VoD service for Eucalyptus platform*", na 2014 *IEEE International Conference on Systems, Man and Cybernetics* (IEEE SMC 2014), San Diego, Califórina/EUA, 2014;

- Maria Bezerra, Rosangela Melo, Jamilson Dantas and Paulo Maciel "*Availability Evaluation of a VoD Streaming Cloud Service*", na 2015 *IEEE International Conference on Systems, Man and Cybernetics* (IEEE SMC 2015), Hong Kong, China, 2015;

- Rosangela Melo, Maria Bezerra, Jamilson Dantas, Rubens Matos, Ivanildo Melo, Paulo Maciel, "*VoD in Eucalyptus Platform: Availability Modeling and Sensibility Analysis*", na *International Conference on Network and Service Management*, 2014, Rio de Janeiro. CNSM 2014, 2014;

- Rosangela Melo, Maria Bezerra, Jamilson Dantas, Rubens Matos, Ivanildo Melo, Paulo Maciel, "*Sensitivity Analysis of Availability of Video Streaming Service in Cloud Computing*", na *IEEE International Performance Computing and Communications Conference*, 2014, Austin. IEEE IPCCC 2014, 2014.

- Rosangela Melo, Maria Bezerra, Jamilson Dantas, Rubens Matos, Ivanildo Melo, Paulo Maciel, "*Redundant VoD Streaming Service in a Private Cloud: Availabi-*

lity Modeling and Sensitivity Analysis", no periódico *Mathematical Problems in Engineering (Print)*, v. 2014, p. 1-14, 2014.

- Rosangela Melo, Maria Bezerra, Jamilson Dantas, Rubens Matos, Ivanildo Melo, Paulo Maciel, "*Availability And Sensitivity Analysis In Video Streaming Services Hosted On Private Clouds*", na *10th Iberian Conference on Information Systems and Technologies* (CISTI'2015), Aveiro/Portugal, 2015;

- Rosangela Melo, Maria Bezerra, Jamilson Dantas, Rubens Matos, Ivanildo Melo, Paulo Maciel, "*Video on Demand hosted in Private Cloud: Availability Modeling and Sensitivity Analysis*", na *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (DSN'2015), Rio de Janeiro/Brazil, 2015;

Além dos trabalhos publicados, temos o seguinte ainda em fase de finalização e/ou aguardando publicação:

- Maria Bezerra, Rosangela Melo, Jamilson Dantas and Paulo Maciel "*Availability modeling and analysis for VoD Service*".

- Maria Bezerra, Rosangela Melo, Jamilson Dantas and Paulo Maciel "*Availability Modeling and Cost Analysis for VoD Streaming Service*".

6.2 Trabalhos futuros

Durante o desenvolvimento do trabalho, identificamos alguns pontos que deveriam ser classificados como extensões do trabalho atual para atividades futuras. Dentre os quais, podemos destacar:

O primeiro ítem para proposta futura, é o de **inclusão de novos serviços ao sistema de vídeo *streaming*.** Um exemplo de um novo tipo de serviço a ser incluído no trabalho, é o *live streaming*, que funciona de forma similar ao *streaming* apresentado neste trabalho, mas que pode servir de análise a fim de observar o comportamento da disponibilidade diante do uso destes serviços. A **Inclusão de mais *VMs* por arquitetura** também é um ponto destacado para proposta futura. Neste trabalho só foi considerado o uso de uma *VM* por arquitetura, sendo sugerida a análise do comportamento da disponibilidade diante do uso de mais de uma *VM* por arquitetura.

A **análise de outras métricas de dependabilidade** também é algo importante a ser trabalhado em variações futuras deste trabalho, visto que a métrica utilizada para análise neste trabalho foi a de disponibilidade, então a análise de outras métricas de dependabilidade como confiança, segurança, integridade e manutenabulidade são pontos que merecem análise em propostas futuras. **Análise de outros mecanismos de replicação:** Os mecanismos de replicação utilizados durante esta pesquisa foram os *warm-stanb* e *hot-stand*. Mesmo assim, o *hot-stand*

só foi considerado na proposta de arquitetura secundária. Desta forma, é sugerida a análise do mecanismo de replicação *cold-stand* em todas as arquiteturas, bem como o *hot-stand*, a fim de observar o comportamento das métricas de dependabilidade utilizadas.

Realização de testes de desempenho e performance nas arquiteturas propostas: Durante o desenvolvimento do trabalho, questões de desempenho e performance não foram analisadas. Desta forma, uma das propostas futuras é a inclusão destes tipos de análise para a proposição de cada arquitetura, identificando o perfil de atendimento de cada cenário proposto, capacidade de atendimento e melhorias na qualidade do serviço prestado, por exemplo. **Planejamento das arquiteturas propostas:** Como algumas arquiteturas foram propostas, é interessante que haja um estudo de planejamento para tais arquiteturas, consideranto custos, *Service Level Agreement (SLA)* e tipos de arquitetura adequados para cada situação.

Referências

ABDALLAH, H.; HAMZA, M. On the sensitivity analysis of the expected accumulated reward. **Performance Evaluation**, [S.l.], v.47, n.2, p.163–179, 2002.

AMAZON. Disponível em `http://aws.amazon.com/pt/`.

APACHE. Disponível em `http://www.apache.org/`.

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.

ARMBRUST, M. et al. A view of cloud computing. **Communications of the ACM**, [S.l.], v.53, n.4, p.50–58, 2010.

ARMBRUST, M. et al. A view of cloud computing. **Communications of the ACM**, [S.l.], v.53, n.4, p.50–58, 2010.

AVIZIENIS, A. et al. **Fundamental concepts of dependability**. [S.l.]: University of Newcastle upon Tyne, Computing Science, 2001.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. **Dependable and Secure Computing, IEEE Transactions on**, [S.l.], v.1, n.1, p.11–33, 2004.

BAUER E; ADAMS, R. E. D. **Beyond Redundancy**. [S.l.]: Institute of Electrical and Electronics Engineers, 2012.

BEZERRA, M. C. et al. Availability Modeling and Analysis of a VoD Service for Eucalyptus Platform. In: SYSTEMS, MAN, AND CYBERNETICS (SMC), 2014 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014.

BLAKE, J. T.; REIBMAN, A. L.; TRIVEDI, K. S. Sensitivity analysis of reliability and performability measures for multiprocessor systems. In: ACM SIGMETRICS PERFORMANCE EVALUATION REVIEW. **Anais...** [S.l.: s.n.], 1988. v.16, n.1, p.177–186.

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

CARDOSO, F. C. Conceitos de rede virtual privada para streaming seguro de vídeo. **Universidade São Franscisco**, [S.l.], 2010.

CAROLAN, J. et al. Introduction to cloud computing architecture. **White Paper, 1st edn. Sun Micro Systems Inc**, [S.l.], 2009.

CAROLAN, J. et al. Introduction to cloud computing architecture. **White Paper, 1st edn. Sun Micro Systems Inc**, [S.l.], 2009.

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

CENTOS. **CentOS Project**. Disponível em: http://www.centos.org/.

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.

CLARK, C. et al. Live migration of virtual machines. In: SYMPOSIUM ON NETWORKED SYSTEMS DESIGN & IMPLEMENTATION-VOLUME 2, 2. **Proceedings...** [S.l.: s.n.], 2005. p.273–286.

DALZIELL, E.; MCMANUS, S. Resilience, vulnerability, and adaptive capacity: implications for system performance. , [S.l.], 2004.

DANTAS, J. et al. An availability model for eucalyptus platform: an analysis of warm-standy replication mechanism. , [S.l.], p.1664–1669, 2012.

DINH, H. T. et al. A survey of mobile cloud computing: architecture, applications, and approaches. **Wireless communications and mobile computing**, [S.l.], v.13, n.18, p.1587–1611, 2013.

DOWNING, D. J.; GARDNER, R.; HOFFMAN, F. An examination of response-surface methodologies for uncertainty analysis in assessment models. **Technometrics**, [S.l.], v.27, n.2, p.151–163, 1985.

DROPBOX. Disponível em https://www.dropbox.com/.

EBELING, C. E. **An introduction to reliability and maintainability engineering**. [S.l.]: Tata McGraw-Hill Education, 2004.

EC2, A. Disponível em http://aws.amazon.com/pt/ec2/.

EDUCAçãO, M. da. **Censo da Educação Superior 2013**. Disponível em http://download.inep.gov.br/educacao_superior/censo_superior/apresentacao/2014/coletiva_censo_superior_2013.pdf.

EJLALI, A. et al. A hybrid fault injection approach based on simulation and emulation co-operation. In: ANNUAL IEEE/IFIP INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS (DSN), 2013. **Anais...** [S.l.: s.n.], 2003. p.479–479.

EJLALI, A.; GHASSEM MIREMADI, S. FPGA-based fault injection into switch-level models. **Microprocessors and Microsystems**, [S.l.], v.28, n.5, p.317–327, 2004.

ELEFTHERIADIS, A.; ANASTASSIOU, D. Meeting arbitrary QoS constraints using dynamic rate shaping of coded digital video. In: NETWORK AND OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO. **Anais...** [S.l.: s.n.], 1995. p.88–100.

ENGINE, G. A. Google Cloud Platform. Disponível em: https://cloud.google.com/appengine/.

EUCALYPTUS. **Open Source Private Cloud Software**. Disponível em https://www.eucalyptus.com/.

EUCALYPTUS. **Eucalyptus**: open source private cloud software. Disponível em https://www.eucalyptus.com/eucalyptus-cloud/iaas.

EUCALYPTUS. **Eucalyptus Components**. Disponível em https://www.eucalyptus.com/docs/eucalyptus/3.2/ig/euca_components.html.

EUCALYPTUS cloud computing platform - administrator guide. Technical report, Eucalyptus Systems, Inc., Version 2.0.

FIGUEIREDO, J. et al. Estimating reliability importance and total cost of acquisition for data center power infrastructures. In: SYSTEMS, MAN, AND CYBERNETICS (SMC), 2011 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.421–426.

FORCE, S. Sales Force Plataform. Disponível em: http://www.salesforce.com/platform/overview/.

FRANK, P. M. **Introduction to System Sensitivity Theory**. [S.l.]: Academic Press Inc., 1978.

GHOSH, R. et al. Scalable analytics for iaas cloud availability. , [S.l.], 2014.

GOGRID. GoGrid cloud hosting. Disponível em: http://www.gogrid.com/.

GOLDBERG, R. P. Survey of virtual machine research. **Computer**, [S.l.], v.7, n.6, p.34–45, 1974.

GONG, C. et al. The characteristics of cloud computing. In: PARALLEL PROCESSING WORKSHOPS (ICPPW), 2010 39TH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.275–279.

GOOGLE. Google Compute Engine. Disponível em https://cloud.google.com/compute/.

GOOGLE. Google Drive. Disponível em: https://drive.google.com/.

GOOGLE. Serviços de e-mail do Google. Disponível em: http://www.gmail.com.

GUIMARÃES, A. et al. Dependability analysis in redundant communication networks using reliability importance. In: THE 2011 INTERNATIONAL CONFERENCE ON INFORMATION AND NETWORK TECHNOLOGY (ICINT 2011)–CHENNAI, INDIA. **Anais...** [S.l.: s.n.], 2011.

HAMBY, D. A review of techniques for parameter sensitivity analysis of environmental models. **Environmental Monitoring and Assessment**, [S.l.], v.32, n.2, p.135–154, 1994.

HEARTBEAT. Linux-HA Project. Disponível em http://www.linux-ha.org.

HERZOG, U. Formal methods for performance evaluation. In: **Lectures on Formal Methods and PerformanceAnalysis**. [S.l.]: Springer, 2001. p.1–37.

HU, T. et al. Mttf of composite web services. In: PARALLEL AND DISTRIBUTED PROCESSING WITH APPLICATIONS (ISPA), 2010 INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2010. p.130–137.

HWANG, I.-S. et al. Scalable architecture for VOD service enhancement based on a cache scheme in an Ethernet passive optical network. **Optical Communications and Networking, IEEE/OSA Journal of**, [S.l.], v.5, n.4, p.271–282, 2013.

J. ARAUJO R. MATOS, P. M. Software Again Issues on the Eucalyptus Cloud Computing Infrastructure. **Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics**, [S.l.], 2011.

JOHNSON, D. et al. **Eucalyptus Beginner's Guide**. 2010.

KEESEE, W. **A Method Of Determing a Confidence Interval For Availability**. [S.l.]: DTIC Document, 1965.

KHAZAEI, H. et al. Availability analysis of cloud computing centers. In: GLOBAL COMMUNICATIONS CONFERENCE (GLOBECOM), 2012 IEEE. **Anais...** [S.l.: s.n.], 2012. p.1957–1962.

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.

KRSUL, I. et al. Vmplants: providing and managing virtual machine execution environments for grid computing. In: SUPERCOMPUTING, 2004. PROCEEDINGS OF THE ACM/IEEE SC2004 CONFERENCE. **Anais...** [S.l.: s.n.], 2004. p.7–7.

KUO, W.; ZUO, M. J. **Optimal reliability modeling**: principles and applications. [S.l.]: John Wiley & Sons, 2003.

LAPRIE, J.-C. **Dependability**: basic concepts and terminology. [S.l.]: Springer, 1992.

LIU, Y.; LEE, J. Y. Providing predictable streaming performance in mobile video streaming. In: COMMUNICATIONS (ICC), 2014 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014. p.2276–2281.

LIVESTREAM. **Cloud service to broadcast your event live**. Disponível em `https://new.livestream.com/platform`.

MACIEL, P. et al. Dependability modeling. In: **Performance and Dependability in Service Computing**: concepts, techniques and research directions. Hershey: IGI Global, 2011.

MALHOTRA, M.; TRIVEDI, K. S. Power-hierarchy of dependability-model types. **Reliability, IEEE Transactions on**, [S.l.], v.43, n.3, p.493–502, 1994.

MANAGER, V. M. Disponível em `http://virt-manager.org/`.

MARINO, S. et al. A methodology for performing global uncertainty and sensitivity analysis in systems biology. **Journal of theoretical biology**, [S.l.], v.254, n.1, p.178–196, 2008.

MATHEMATICA, W. **Mathematica**. Disponível em: `http://www.wolfram.com/mathematica/`.

MATOS JUNIOR, R. et al. Sensitivity analysis of availability of redundancy in computer networks. In: CTRQ 2011, THE FOURTH INTERNATIONAL CONFERENCE ON COMMUNICATION THEORY, RELIABILITY, AND QUALITY OF SERVICE. **Anais...** [S.l.: s.n.], 2011. p.115–121.

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.

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.

MCCANNE, S.; JACOBSON, V.; VETTERLI, M. Receiver-driven layered multicast. **ACM SIGCOMM Computer Communication Review**, [S.l.], v.26, n.4, p.117–130, 1996.

MELO, M. D.; MACIEL, P. R. M. O. Modelos de Disponibilidade Para Nuvens Privadas: rejuvenescimento de software habilitado por agendamento de migração de vms. , [S.l.], 2014.

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.

MERCURY. **Mercury Tool**. Disponível em: `https://sites.google.com/site/mercurytooldownload/`.

MICROSOF. Windows Azure. Disponível em: `http://azure.microsoft.com/pt-br/`.

MINITAB. **Mais informações**. Disponível em `http://www.minitab.com/pt-br/`.

MODCS. **Obtenção da Fórmula Fechada**. Disponível em: `http://www.modcs.org/wp-content/uploads/2015/01/Tutorial-Mathematica-F%C3%B3rmula-Fechada-1.pdf`.

MUSA, J. D. **Software reliability engineering**: more reliable software, faster and cheaper. [S.l.]: Tata McGraw-Hill Education, 2004.

NETFLIX. Disponível em `https://www.netflix.com/`.

NETFLIX. **Company Overview**. Disponível em `https://pr.netflix.com/WebClient/loginPageSalesNetWorksAction.do?contentGroupId=10476&contentGroup=Company+Facts`.

NIMBUS. **Nimbus is cloud computing for science.** Disponível em `http://www.nimbusproject.org/`.

OPENNEBULA. **A solution to build clouds and manage data center virtualization**. Disponível em `http://opennebula.org/`.

PARHAMI, B. From defects to failures: a view of dependable computing. **ACM SIGARCH Computer Architecture News**, [S.l.], v.16, n.4, p.157–168, 1988.

PWC. **Mercado de entretenimento e mídia brasileiro chegará a US$ 71 bilhões em 2017**. Disponível em `https://www.pwc.com.br/pt/sala-de-imprensa/assets/press-release/release-14th-global-entertainment-media-outlook-2013-2017-06-13.pdf`.

RESNICK, R. I. **A modern taxonomy of high availability**. [S.l.]: Technical report, Interlog, 1996.

SERVICE, A. S. S. **Armazenamento online na nuvem para dados e arquivos**. Disponível em `http://aws.amazon.com/pt/s3/`.

SILVA, B. et al. ASTRO: an integrated environment for dependability and sustainability evaluation. **Sustainable Computing: Informatics and Systems**, [S.l.], v.2, p.1–31, 2012.

SOUSA, E. T. G. de. Avaliação do Impacto de uma Política de Manutenção na Performabilidade de Sistemas de Transferência Eletrônica de Fundos. , [S.l.], 2009.

STANDART, N. I. of; TECHNOLOGY'S. **Cloud Computing**. Disponível em `http://www.nist.gov`.

SUN, D. et al. A dependability model to enhance security of cloud environment using system-level virtualization techniques. In: PERVASIVE COMPUTING SIGNAL PROCESSING AND APPLICATIONS (PCSPA), 2010 FIRST INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.305–310.

TAN, W.-T.; ZAKHOR, A. Real-time Internet video using error resilient scalable compression and TCP-friendly transport protocol. **Multimedia, IEEE Transactions on**, [S.l.], v.1, n.2, p.172–186, 1999.

TRIVEDI, K. S. et al. Dependability and security models. In: DESIGN OF RELIABLE COMMUNICATION NETWORKS, 2009. DRCN 2009. 7TH INTERNATIONAL WORKSHOP ON. **Anais...** [S.l.: s.n.], 2009. p.11–20.

TRUONG, H.-L.; DUSTDAR, S. On analyzing and specifying concerns for data as a service. In: SERVICES COMPUTING CONFERENCE, 2009. APSCC 2009. IEEE ASIA-PACIFIC. **Anais...** [S.l.: s.n.], 2009. p.87–94.

UBUNTU. **Ubuntu Server Overview**. Disponível em: `http://www.ubuntu.com/server`.

USTREAM. **The Video Platform**. Disponível em `http://www.ustream.tv/`.

VERA-SSIMO, P.; RODRIGUES, L. **Distributed systems for system architects**. [S.l.]: Springer, 2001. v.1.

VIDEOLAN. Disponível em `http://www.videolan.org/vlc/index.html`.

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

WAN, Y. et al. The adaptive heartbeat design of high availability RAID dual-controller. In: MULTIMEDIA AND UBIQUITOUS ENGINEERING, 2008. MUE 2008. INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2008. p.45–50.

XIN, W.; SCHULZRINNE, H. Comparison of adaptive Internet multimedia applications. **IEICE Transactions on Communications**, [S.l.], v.82, n.6, p.806–818, 1999.

YOUTUBE. **YouTube**. Disponível em `https://www.youtube.com/`.

YOUTUBE. **Estatísticas**. Disponível em https://www.youtube.com/yt/press/pt-BR/statistics.html.

YOUTUBE. **Busca pelo termo "aula"**. Disponível em https://www.youtube.com/results?search_query=aula.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. **Journal of internet services and applications**, [S.l.], v.1, n.1, p.7–18, 2010.

ZHANG, Q. et al. Robust scalable video streaming over Internet with network-adaptive congestion control and unequal loss protection. In: PACKET VIDEO WORKSHOP. **Proceedings...** [S.l.: s.n.], 2001.

ZHAO, M.; FIGUEIREDO, R. J. Experimental study of virtual machine migration in support of reservation of cluster resources. In: VIRTUALIZATION TECHNOLOGY IN DISTRIBUTED COMPUTING, 2. **Proceedings...** [S.l.: s.n.], 2007. p.5.

ZHU, W. et al. Multimedia cloud computing. **Signal Processing Magazine, IEEE**, [S.l.], v.28, n.3, p.59–69, 2011.

Apêndice

A

Ferramenta Mercury

Para a obtenção dos modelos desta pesquisa e suas respectivas análises, fizemos uso da ferramenta Mercury (SILVA et al., 2012; MERCURY, 2014), específica para modelagem analítica. É interessante fornecer um espaço para apresentar essa ferramenta, visando justificar a sua escolha.

A ferramenta Mercury é o *kernel* de outra ferramenta denominada ASTRO (FIGUEIREDO et al., 2011). A ferramenta ASTRO é um ambiente integrado para a avaliação de sustentabilidade, custo e disponibilidade de ambientes de *data center*. Esta ferramenta permite que usuários modelem sistemas de TI, sistemas energéticos e sistemas de resfriamento, sem a necessidade de conhecer os formalismos matemáticos que serão empregados para realizar a análise das propriedades desses sistemas.

O Mercury, por sua vez, além de ser o *kernel* de modelagem e avaliação dos modelos de alto nível criados pelo ASTRO, também é uma ferramenta para a criação e análise de modelos de RBD, cadeias de Markov, redes de Petri estocásticas e modelos de fluxo energético (EFM). As características de funcionamento da ferramenta Mercury são vistas na Figura A.1 a seguir:

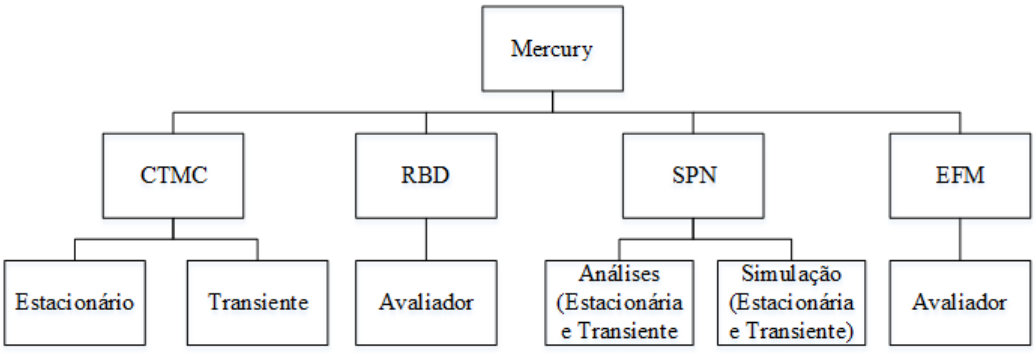


Figura A.1: Características da Ferramenta Mercury

B

Script de Monitoramento do Apache

```
1  echo "Fault injection Apache...."
2  echo "# Event GT Date" >> ApacheFault.log
3
4  C1=0
5  while [ true ] ;
6  do
7          C1=$(($C1+1))
8          STATUSS=$(nmap -p 80 192.168.10.50 | grep 80/tcp | awk '{
               print $2}')
9          SSTATUS=$(nmap -p 80 192.168.10.50 | grep "Nmap done" | awk
               '{print $6}')
10
11     if [ "$SSTATUS" == "(1" ] ; then
12
13       if [ "$STATUSS" == "open" ] ; then
14          TIME=$(Rscript expFailApache.r | awk '{print $2}')
15          sleep $TIME
16          ./ApacheFailure.sh
17          echo "ok-F" >> ApacheFault.log
18          echo "Fault injected"
19          echo  $C1 "Fault" $TIME $(date | awk '{ print $2, $3, $4 }')
               >> ApacheFault.log
20
21       fi
22
23       if [ "$STATUSS" == "closed" ] ; then
24          TIMER=$(Rscript expRepairApache.r | awk '{print $2}')
25          sleep $TIMER
26          ./ApacheRepair.sh
27          echo "ok-R" >> ApacheFault.log
28          echo "Repair Injected"
29          echo $C1 "Repair" $TIMER $(date | awk '{ print $2, $3, $4 }')
                 >> ApacheFault.log
30       fi
```

```
31      else echo "VM desligada"
32     fi
33  done
34  echo "Terminou Apache"
```

C

Scripts de Monitoramento do VLC

```
1  echo "Fault injection Apache...."
2  echo "# Event GT Date" >> VLCFault.log
3  C1=0
4  while [ true ] ;
5  do
6          C1=$(($C1+1))
7          STATUSS=$(nmap -p 8085 192.168.10.50 | grep 8085/tcp | awk '{
               print $2}')
8          SSTATUS=$(nmap -p 80 192.168.10.50 | grep "Nmap done" | awk
               '{print $6}')
9
10    if [ "$SSTATUS" == "(1" ] ; then
11
12      if [ "$STATUSS" == "open" ] ; then
13         TIME=$(Rscript expFailVLC.r | awk '{print $2}')
14         sleep $TIME
15         ./VLCFailure.sh
16         echo "ok-F" >> VLCFault.log
17         echo "Fault injected"
18         echo  $C1 "Fault" $TIME $(date | awk '{ print $2, $3, $4 }')
               >> VLCFault.log
19      fi
20
21      if [ "$STATUSS" == "closed" ] ; then
22         TIMER=$(Rscript expRepairVLC.r | awk '{print $2}')
23         sleep $TIMER
24         ./VLCRepair.sh
25         echo "ok-R" >> VLCFault.log
26         echo "Repair Injected"
27         echo $C1 "Repair" $TIMER $(date | awk '{ print $2, $3, $4 }')
                 >> VLCFault.log
28      fi
29     else echo "VM Off"
30    fi
```

```
31 done
32 echo "Terminou VLC"
```

D

Script de Monitoramento da VM

```
1  #!/bin/bash
2
3  #******************************** FUNCOES
       ***********************************
4
5  matarVMs(){
6     instancias=`euca-describe-instances | grep INSTANCE | grep running
          | awk '{print $2}'`
7     euca-terminate-instances $instancias
8  }
9
10 verificaVmUp(){
11    booting=TRUE
12    while [ $booting = TRUE ]
13    do
14      running=`euca-describe-instances | grep running | wc -l`
15      if [ $running -gt 0 ]
16      then
17        booting=FALSE
18      fi
19   done
20   echo "A VM est  Running!" >> log-tempos-vm.txt
21   echo "A VM est  Running!"
22 }
23
24 instanciaVMs(){
25     euca-run-instances -t m1.small -z CLUSTER01 emi-5A003F00
26     verificaVmUp
27     sleep 12
28     ./Mountvol.sh
29     sleep 3
30     ./sstart.sh
31     sleep 5
32 }
```

```
33
34 #****************************** Inje  o de falha na VM
       ***********************************
35
36 while [ true ] ;
37 do
38
39         running=`euca-describe-instances | grep running | wc -l`
40
41      if [ "$running" = 1 ] ; then
42      TIME=$(Rscript expFailVM.r | awk '{print $2}')
43         sleep $TIME
44         matarVMs
45         echo "ok-F" >> VMFault.log
46         echo "Fault injected"
47         echo  "Fault" $TIME $(date | awk '{ print $2, $3, $4 }') >>
               VMFault.log
48      fi
49
50      if [ "$running" = 0 ] ; then
51      TIMER=$(Rscript expRepairVM.r | awk '{print $2}')
52         sleep $TIMER
53         instanciaVMs
54         echo "ok-R" >> VMFault.log
55         echo "Repair Injected"
56         echo "Repair" $TIMER $(date | awk '{ print $2, $3, $4 }')  >>
               VMFault.log
57      fi
58
59 done
60
61 echo "Terminou VM"
```