Pós-Graduação em Ciência da Computação

"Modelos para Análise de Dependabilidade de Arquiteturas de Computação em Nuvem"

Por

Jamilson Ramalho Dantas

Dissertação de Mestrado

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

RECIFE, Março/2013

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

Jamilson Ramalho Dantas

"Modelos para Análise de Dependabilidade de Arquiteturas de Computação em 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: *Prof. Dr. Paulo Romero Martins Maciel*

RECIFE, Março/2013

Eu dedico essa dissertação para minha família que é a base
para meu crescimento profissional.

Agradecimentos

Agradeço a Deus por permitir e favorecer, de todas as formas, a realização e a conclusão deste trabalho. Por me iluminar e tornar claras as decisões que tomei nesse período.

Agradeço aos meus pais e aos meus irmãos, que sempre me apoiaram, principalmente nos momentos de maior dificuldade, pelas horas de preocupação.

Agradeço a Renata Dantas, minha esposa e companheira de todas as horas, pelo incentivo, força, apoio, auxílio e por entender minha ausência em alguns momentos dessa jornada, possibilitando que o esforço fosse recompensando com a concretização deste trabalho. Agradeço também a minha querida sogra Iraci de Sá, que me deu todo apoio e suporte no momento do nascimento de meu filho Davi, deixando-me menos "sufocado"para a escrita do trabalho. Agradeço ainda a Neila Raquel, por corrigir a escrita de minha dissertação.

Aos amigos Rubens Matos, Jean Teixeira, Rafael, João e tantos outros que me apoiaram de alguma forma para que fosse possível a conclusão deste trabalho. Agradeço ao meu orientador, professor Paulo Maciel, que sempre esteve disponível para me orientar, confiou no meu trabalho e me incentivou, além de me acolher, fazendo com que eu me integrasse ao grupo MoDCS mesmo sem me conhecer.

Agradeço ainda a todos aqueles que de alguma forma contribuiram para a realização deste trabalho, incentivando e apoiando.

Conhecimento do que é possível é o começo da felicidade.

—GEORGE SANTAYANA

Resumo

A tecnologia avança gradativamente à medida que estudos e pesquisas são realizadas. Com isso, a necessidade de alto poder de computação e segurança dos dados é exigida por usuários comuns, haja vista que equipamentos eletrônicos, sistemas de software, hardware e meios de interconexões (cabos, *wireless*) tendem a falhar, tornando-se indisponíveis por tempo indeterminado. A indisponibilidade pode ser acometida por falhas, defeitos ou manutenções planejadas, troca de equipamentos ou atualizações de *software*, tornando-se um desafio para empresas provedoras de serviços em nuvem. Alta disponibilidade em serviços de computação em nuvem é essencial para manter a confiança do cliente e evitar perdas de receita devido a penalidades de violação de SLA's (acordo de níveis de serviço). Uma vez que os componentes de software e hardware de infraestruturas de nuvem podem ter confiabilidade limitada, mecanismos de tolerância a falhas são meios para atingir os requisitos de confiabilidade necessárias. Este trabalho investiga os benefícios de um mecanismo de replicação *warm-standy* em um ambiente de computação em nuvem Eucalyptus. Uma abordagem de modelagem hierárquica heterogénea é usada para representar as arquiteturas e avaliar a disponibilidade das infraestruturas. Falhas de hardware e de software são considerados nos modelos analíticos propostos. Estes modelos também são utilizados para a obtenção de equações para calcular a disponibilidade da infraestrutura de nuvem. Para demonstrar a aplicabilidade deste trabalho foram gerados três estudos de caso.

Palavras-chave: Computação em Nuvem, Disponibilidade, Modelos Analíticos, Confiabilidade.

Abstract

High availability computing power and data security are required by many computer systems, considering that, electronic equipment, systems software, hardware and means of interconnections (cables, wireless) can to fail. The availability may be affected by faults, defects or planned maintenance, exchange of equipment or software upgrades, becoming a challenge for companies providing cloud services. High availability in cloud computing services is essential for maintaining customer confidence and avoiding revenue losses due to SLA violation penalties, since cloud services are based on the "pay-as-you-go"model. The software and hardware components of cloud infrastructures may have limited reliability, fault tolerance mechanisms are means of achieving the necessary dependability requirements. This work investigates the benefits of a warm-standy replication mechanism in a Eucalyptus cloud computing environment. A hierarchical heterogeneous modeling approach is used to represent the architectures and compare their availability characteristics. These models are also used to obtain closed-form equations for computing the availability of the cloud infrastructure. Three cases studies demonstrate the applicability of this work.

Keywords: Cloud Computing, Availability, Analytical Models, Reliability.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

API *Application Programming Interface*

AWS *Amazon Web Services*

CC Controlador de *Cluster* - *Cluster Controller*

CLC Controlador da Nuvem - *Cloud Controller*

CTMC *Continuous-time Markov chains*

DRBD *Distributed Replicated Block Device*

DTMC *Discrete-time Markov chains*

EBS *Elastic Block Store*

EC2 *Amazon Elastic Compute Cloud*

IaaS *Infrastructure as a Service*

KVM *Kernel-based Virtual Machine*

MRM *Markov Reward Model*

MTTF *Mean Time to Failure*

MTTR *Mean Time to Repair*

NC Controlador do Nó - *Node Controller*

PaaS *Platform as a Service*

PN *Petri Nets*

RAID *Redundant Array of Independent Disks*

RBD *Reliability Block Diagram*

RdP Rede de Petri

S3 *Simple Storage Service*

SaaS *Software as a service*

SC Controlador de Armazenamento - *Storage Controller*

SLA *Service Level Agreement*

VM *Virtual Machine*

VMM *Virtual Machine Monitor*

1 Introdução

A única exigência que faço aos meus leitores é que devem dedicar as suas vidas à leitura das minhas obras.

—JAMES JOYCE

Sistemas computacionais, bem como seus componentes de *software*, *hardware* e meios de interconexões (cabos, *wireless*) tendem a falhar inesperadamente, tornando o sistema indisponível por tempo indeterminado.

Em sistemas que necessitem de alta disponibilidade e confiabilidade, é necessário criar métodos para a detecção e correcção de erros na tentativa de evitar possíveis falhas. Técnicas para alcançar altos níveis de dependabilidade vêm sendo aprimoradas a fim de contornar falhas e alcançar maior tempo de atividade e segurança da informação.

A indisponibilidade é um dos problemas mais comuns que os administradores de sistemas devem solucionar com rapidez e agilidade, visto que afeta diretamente os usuários finais dos sistemas [73]. É cada vez mais comum a utilização de meios eletrônicos e recursos computacionais em empresas de diferentes portes. A fim de disponibilizar serviços por maior tempo possível, tais empresas tentam adotar medidas de segurança e plano de tolerância a falhas na tentativa de evitar prejuízos.

A necessidade de poder de processamento e utilização de recursos computacionais está presente nas empresas. Em consequência, soluções estão sendo criadas para que essas necessidades sejam supridas. O paradigma de computação em nuvem está sendo usado para compartilhar recursos computacionais bem como *software* e aplicativos em geral.

Sistemas em nuvem utilizam técnicas de virtualização onde usuários podem fazer uso de máquinas virtuais com configurações quaisquer [6], dependendo dos recursos

computacionais disponíveis nas máquinas físicas.

Eucalyptus, OpenNebula e Nimbus [30] [57] [56] são três ferramentas *open-source* conhecidas para plataforma de computação em nuvem. A função geral desses sistemas é gerenciar o provisionamento de máquinas virtuais para uma nuvem, fornecendo infraestrutura como serviço (IaaS) [7] [66].

A idéia de utilizar a nuvem como provedor de serviços é uma solução para que necessidades por recursos computacionais sejam minimizadas, fornecendo poder de processamento aos usuários. Várias empresas oferecem esses serviços no mercado: *Amazon, Sun Cloud, IBM SmartCloud, Azure* [2] [58] [44] [10].

Empresas que utilizam tais serviços requerem disponibilidade, segurança e confiabilidade dos dados. Por outro lado, as empresas que oferecem tais serviços têm que garantir a entrega total do serviço por meio de acordos de níveis de serviços (SLA's). Assim, avaliação de dependabilidade [8] é uma atividade essencial para promover a melhoria da qualidade do serviço prestado e para o planejamento de infraestruturas de sistemas de computação. Ressalte-se, porém, que falhas em sistemas são inevitáveis, mas podem ser contornadas por meios de técnicas adequadas.

Muitas técnicas têm sido propostas e adotadas para construção de *cluster de failover* [49]. Muitas técnicas são baseadas em redundância, isto é, a replicação do componente de modo que várias máquinas trabalhem para um fim comum e garantam a segurança dos dados e a disponibilidade mesmo em caso de falha de algum componente.

A importância da computação em nuvem tem sido cada vez maior dentro de uma grande proporção de empresas de TI em todo o mundo. Plataformas de software, tais como *Eucalyptus* [30], fornecem meios para a construção de nuvens privadas e híbridas no estilo de Infraestrutura como Serviço, em cima dos equipamentos de *hardware* instalados no local [30].

Mecanismos de tolerância a falhas, tais como técnicas de replicação, são abordagens importantes para lidar com confiabilidade limitada de software e hardware. Uma vez que se pretende, com o uso de aplicativos baseados em nuvem, ampla acessibilidade, ou seja, ser acessível em qualquer lugar e a qualquer hora, a confiabilidade torna-se ainda mais importante e ainda mais difícil de alcançar em tais ambientes [67].

Essa dissertação investiga os benefícios do mecanismo de replicação *warm-standby* [38][53] em um ambiente de computação em nuvem *Eucalyptus*, considerando várias arquiteturas, em relação ao números de *cluster* e seus subsistemas redundantes. Uma abordagem de modelagem heterogênea hierárquico é usada para representar arquiteturas redundantes e comparar a sua disponibilidade, sendo também considerado o custo de

aquisição de computadores. Tanto falhas de *hardware* como de *software* são considerados em nossos modelos analíticos. Estes modelos também são utilizados para a obtenção de equações de forma fechada a fim de calcular a disponibilidade das infraestruturas da nuvem. Os estudos de caso são realizados através dos modelos hierárquicos.

1.1 Motivação e Justificativa

A busca por confiabilidade, disponibilidade e segurança da informação é cada vez maior em empresas que buscam oferecer serviços de forma ininterrupta.

Softwares têm sido criados com o intuito de prover técnicas de virtualização e fornecer serviços em nuvem, na tentativa de obter confiabilidade dos dados, alta disponibilidade, segurança, economia de energia, entre outros. O *Eucalyptus* é uma dessas soluções, feito para construção de nuvens privadas e híbridas, ela fornece infraestrutura como serviço - IaaS, permitindo o controle de grandes coleções de recursos [27][30][28][29]. Ainda assim, a confiabilidade desse tipo de sistema pode ser afetada por falhas ou atualizações de *software*, manutenções planejadas e troca de equipamentos.

Determinadas arquiteturas de computação em nuvem possuem componentes que requerem maiores cuidados por serem pontos únicos de entrada de clientes. A redundância desse componente pode ser a solução para alcançar a dependabilidade desejada, porém faz-se necessária a avaliação de outros equipamentos que compõem determinada arquitetura. Pois o uso errado de redundância de componentes pode aumentar o custo com aquisição de computadores, gerando possíveis prejuízos. Modelagem analítica vem sendo bastante utilizada para availar os mais variados tipos de sistemas [38] [45], sejam estes computacionais ou não. A avaliação de dependabilidade por meio de modelos analíticos e de simulação é um meio viável para propor melhorias aos sistemas de computação em nuvem.

O estudo que guia essa dissertação permite que administradores e gestores de tecnologia da informação possam construir arquiteturas em nuvem que forneçam um maior tempo de atividade, considerando disponibilidade de serviços e orientada a capacidade (COA), além do custo de aquisição de máquinas.

1.2 Trabalhos Relacionados

Alguns trabalhos têm sido realizados com diferentes abordagens e técnicas de modelagem com o objetivo de avaliar sistemas computacionais, verificando sua disponibilidade e dependabilidade. A maioria das abordagens utilizadas para avaliação de dependabilidade

de sistemas na nuvem não trata da influência de adição de novos equipamentos. Os trabalhos descritos a seguir apresentam temas relacionados ao desta pesquisa.

Wei *at al.*[75] analisa *Data Centers* virtuais em nuvem, focando em dois atributos de dependabilidade: confiabilidade e disponibilidade dos serviços prestados. No estudo de caso, os autores fazem uso de técnicas de modelagem hierárquica e desenvolvem modelos combinando técnicas de modelagem, como *Reliability block diagram* (RBD)[61] e *Stochastic Petri Nets* (SPN)[31], analisando a relação entre disponibilidade e a carga de trabalho para *data centers* virtuais de computação em nuvem.

Em [47] [48], os autores apresentam um estudo sobre previsão de confiabilidade para servidores baseados em sistema de *cluster* de alta disponibilidade conhecido como OSCAR *(Open Source Cluster Application Resources)*. Os autores prevêem a confiabilidade e a disponibilidade do sistema através de redes de recompensa estocásticas (SRN).

Hong *at al.* [42] usa cadeia de Markov de tempo contínuo (*Continuous Time Markov Chains - CTMC*) para analisar a disponibilidade de sistemas em *cluster* com múltiplos nós. Sua análise considera tanto a falha de modo comum (*Common Mode Failure - CMF*) e a não CMF para os nós do *cluster*. Em primeiro lugar, é analisada a disponibilidade de um sistema com um único nó de *cluster*. Então, quatro modelos de cadeia de Markov de tempo contínuo (CTMC) foram propostos para análise de sistemas com dois e três nós de *cluster*. Resultados numéricos de disponibilidade do sistema foram obtidas a fim de comparar as arquiteturas estudadas.

Kishor S. Trivedi *at al.* [70] apresenta modelagens de alta disponibilidade em plataformas de transmissão de voz e dados usando diagrama de blocos de confiabilidade e cadeias de Markov. Os autores apresentam os modelos com o intuito de avaliar a disponibilidade do sistema e comparar os resultados obtidos utilizando a ferramenta *SHARPE*[64]. Além de realizar uma análise de sensibilidade para identificar os efeitos dos parâmetros críticos do sistema.

Já em [45] os autores propõem modelagem de disponibilidade de sistemas virtualizados. Para isso, foi construído dois *hosts*, um com sistema virtualizado e outro com sistema não virtualizado, uma abordagem hierárquica com árvores de falha para representar o sistema em alto nível e uma abordagem homogênea em cadeias de Markov de tempo contínuo (CTMC) é usado para representar o modelo do sistema em baixo nível. É considerado falhas de *hardware* e *software* incluindo o *Virtual Machine Monitor* (VMM), *Virtual Machine* (VM), e falhas de aplicações. Como resultado os autores analisam a disponibilidade em estado estacionário, disponibilidade orientada a capacidade além do tempo de inatividade anual para as arquiteturas analisadas

Em seguida, com base na análise primária, contrapor ao sistema de *cluster* com dois nós e três, quatro nós de tempo contínuo Markov Chain (CTMC) modelos, no caso de n CMF e, no caso de CMF foram construídas respectivamente. Além disso, a comparação numérica de disponibilidade constante para estes dois casos foi realizada através da resolução de equações lineares e relacionados a conclusão sobre a comparação prioridade e inferioridade de diferentes casos é derivado desse modo, que oferece base teórica para o estabelecimento de sistemas de cluster.

Callou *at al.* [12] propõe um conjunto de modelos para integrar a quantificação do impacto da sustentabilidade, custo e confiabilidade de infra-estruturas de refrigeração de *data center* e apresenta um estudo de caso que analisa o impacto ambiental, as métricas de confiabilidade, bem como os custos de aquisição e operacionais de arquiteturas de refrigeração de cinco *data centers* do mundo real.

Já em [17], os autores propõem um ambiente de nuvem privada adequado para fins de governo eletrônico (*e-government*) e oferecem modelos hierárquicos para descrever a proposta de uma arquitetura com base Eucalyptus e calcular seus resultados de disponibilidade. O modelo hierárquico proposto em [17] utiliza cadeias de Markov distintas para o nível de *cluster* e nível do nó, enquanto o nível de operação é descrita de uma maneira não-formal.

Diferente dos trabalhos anteriores, este trabalho propõe uma análise de dependabilidade em infraestruturas de computação em nuvem por meios de modelagem hierárquica utilizando modelos em RBD e CTMC, considerando questões de disponibilidade em estado estacionário, custo e disponibilidade orientada à capacidade.

Apesar dos trabalhos aqui apresentados mostrarem avaliação de dependabilidade em diferentes ambientes nenhum trata de infraestruturas de computação em nuvem. Os modelos propostos permitem uma visualização de diversos ambientes de nuvem, comparando resultados e mostrando possíveis soluções de construção de uma nuvem privada.

1.3 Objetivos

O principal objetivo deste trabalho é a proposição de um conjunto de modelos para avaliação de dependabilidade das infraestruturas de Computação em Nuvem baseadas no sistema *Eucalyptus*. Esse trabalho auxilia o planejamento das infraestruturas da computação nas nuvens por meio de uma abordagem que faz uso de modelagem hierárquica. De forma mais específica, para se avaliar as infraestruturas de computação nas nuvens o

presente trabalho se propõe a:

- Proppor modelos de dependabilidade para avaliar as infraestruturas de computação em nuvem baseadas no *Eucalyptus*;
- Propor uma metodologia hierárquica que faz uso dos fatores positivos das Cadeias de Markov e RBDs;
- Desenvolver modelos adequados para avaliar a disponibilidade orientada à capacidade das infraestruturas de nuvem.

1.4 Estrutura da Dissertação

O Capítulo 2 apresenta toda a fundamentação teórica do trabalho, bem como conceitos de computação em nuvem, dependabilidade e algumas técnicas para a avaliação de dependabilidade por meio de modelagem de sistemas, que incluem redes de Petri estocásticas, cadeia de Markov e diagrama de blocos de confiabilidade. Em seguida, são apresentados métodos para alta disponibilidade em *cluster*, mostrando a redundância como solução para aquisição de melhoria na disponibilidade. Essa explanação é essencial para melhor entendimento desse trabalho. O Capítulo 3 apresenta o Eucalyptus como solução de construção de arquitetura para computação em nuvem, mostrando sua arquitetura e funcionamento. O Capítulo 4 apresenta os modelos propostos, modelos para arquitetura redundante e não redundante. O Capítulo 5 apresenta os estudos de caso, baseados nos modelos propostos, para avaliação de diferentes arquiteturas redundantes ou não, verificando o acréscimo da disponibilidade para as arquiteturas e custo com aquisição de máquinas. Por fim, o Capítulo 6 apresenta as conclusões obtidas com o desenvolvimento deste trabalho, assim como as principais contribuições e, posteriormente, são apresentadas propostas para trabalhos futuros.

2

Fundamentação Teórica

A mente que se abre a uma nova idéia
jamais volta ao seu tamanho original.
—ALBERT EINSTEIN

Este capítulo apresenta uma introdução sobre computação nas nuvens. Em seguida, apresenta os conceitos básicos sobre dependabilidade. Posteriormente, traz algumas técnicas para avaliação de dependabilidade, incluindo técnicas de modelagens como cadeia de markov de tempo contínuo (CTMC) e diagrama de bloco de confiabilidade (RBD).

2.1 Computação em Nuvem

A computação em nuvem é um modelo de computação onde recursos computacionais tais como poder de processamento, rede, armazenamento e *software* são oferecidos na internet e podem ser acessados remotamente [6]. 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 de qualquer lugar do mundo.

Uma definição formal de computação em nuvem pode ser retirada de *National Institute of Standards and Technology - NIST*, Instituto Nacional de Padrões e Tecnologia dos Estados Unidos da América, onde se diz 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 esforço de gerenciamento mínimo ou interação com o provedor de serviços [54].

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 [21].

- Escalabilidade: Em um sistema de nuvem, recursos de computação são altamente escaláveis, ou seja, possuem capacidade de escala para adição de recursos ou desativar os mesmos, em resposta à evolução das necessidades da aplicação do usuário [29].
- Segurança: Acessos realizados a aplicativos devem ser realizados apenas por pessoas autorizadas; usuários devem confiar seus dados à empresas que fornecem os serviços [13].
- Disponibilidade Estacionária: Espera-se que *sites* de relacionamentos, revistas e jornais eletrônicos ou qualquer sistema computacional sejam disponíveis na ordem de 24x7, ou seja, vinte e quatro horas por dia e sete dias por semana durante um ano [13]. Assim, defini-se disponibilidade do sistema como a probabilidade de que o dispositivo esteja disponível sempre que necessário [46]. Os usuários requerem funcionamento a qualquer hora e em qualquer lugar.
- Confiabilidade: A confiabilidade é definida como a probabilidade de que um dispositivo irá desempenhar as suas funções pretendidas satisfatoriamente durante um período especificado de tempo, sob condições de operação específicas. Com base nesta definição, a confiabilidade é medida como uma probabilidade [46]. Diferente da disponibilidade, a confiabilidade é definida em termos de um intervalo de tempo em vez de um instante de tempo [13].

2.1.1 Modelo 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) [34] [30].

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* [35], *Amazon EC2* [3] e *GoGrid* [33].

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* [4] e *Windows Azure* [19].

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 pode ser acessado na rede por diversos clientes [24]. 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 Docs* [36], *Sales Force* [65] e serviços de e-mail como o *Gmail* [37].

2.1.2 Tipos de Nuvens

Há muitas questões a serem consideradas quando se trata de serviços oferecidos pelas empresas provedoras de serviços em nuvem. Enquanto algumas prestadoras de serviços estão interessadas em reduzir o custo de operação, outras podem preferir alta confiabilidade e segurança de dados [77]. Os tipos de nuvens (ver Figura 2.1) podem ser descritos quanto à natureza de acesso e controle em relação ao uso e provisionamento de recursos físicos e virtuais. Assim, há três tipos de nuvens, cada uma com suas vantagens e desvantagens:

Nuvens Privadas - Esse tipo de implantação da nuvem é restrita apenas a empresas que adotam tal recurso, ficando assim a responsabilidade da mesma em manter seus serviços funcionando. A empresa em questão possui a infraestrutura e tem controle sobre como os aplicativos são implantados nela [13]. Nesse ponto de vista, a empresa possui um alto grau de segurança, confiabilidade e controle da nuvem.

Nuvens públicas - Empresas prestadoras de serviços oferecem recursos para o público em geral. Na nuvem pública, recursos de computação são fornecidos dinamicamente através da Internet por meio de aplicações Web ou serviços da Web, a partir de um fornecedor externo [32]. Esse tipo de nuvem é instalada em locais fora das instalações dos clientes, de maneira que custos e riscos do cliente são minimizados. Assim, um dos benefícios desse tipo de nuvem é que elas podem ser muito maiores do que as

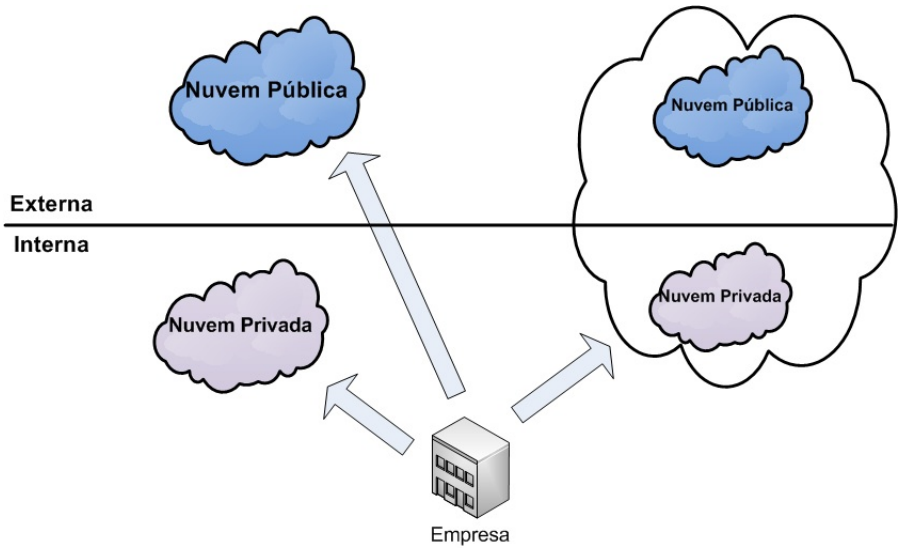


Figura 2.1: Tipos de Nuvens
Adaptado de [32]

nuvens privadas, oferecendo maior capacidade de ajuste à demanda e transferindo a responsabilidade de gerenciamento de infra-estrutura para o provedor da nuvem [13].

Nuvens Híbridas - Esse tipo de modelo é a junção dos dois modelos mencionados anteriormente, onde tanto a empresa fornecedora do serviço quanto o usuário são responsáveis por manter o serviço em funcionamento, configurando-o de tempos em tempos. Uma nuvem híbrida também pode ser usada para lidar com picos de carga de trabalho planejados. Nuvens híbridas oferecem mais flexibilidade do que nuvens públicas e privadas. Especificamente, eles provêem maior controle e segurança sobre os dados e aplicativos em comparação com as nuvens públicas, enquanto ainda facilitam a expansão sob demanda e contração de serviços [77].

2.1.3 Segurança em Nuvem

Confiabilidade e segurança da informação são propriedades muito importantes para um sistema de nuvem que oferece serviços com os recursos de alta disponibilidade, alta escalabilidade, tolerância a falhas e extensibilidade dinâmica [67]. Assim, questões como:

- Será que meus dados serão acessados e copiados por outros?
- Existe equipe capacitada para desenvolver o trabalho desejado?
- Ataques entre máquinas virtuais podem ser evitados?

Essas questões são abordadas quando se trata de dados confidenciais e requerem altos níveis de segurança da informação.

Nos *Data Centers* tradicionais, as abordagens comuns para a segurança da informação incluem *firewall*, zonas desmilitarizadas, segmentação de rede, detecção de intrusão, sistemas de prevenção e ferramentas de monitoramento de rede [32]. Na computação em nuvem, esses tipos de abordagens são requeridos, talvez com os mesmos cuidados ou até cuidados maiores por serem sistemas de fácil escalabilidade e altamente distribuído geograficamente, além do grande uso nos últimos anos.

Esse tipo de serviço pode ser facilmente resolvido em se tratando de servidores físicos, no entanto serviços em nuvem trabalham por meio de sistemas virtualizados. Assim, a segmentação física e segurança baseada em hardware provavelmente não protegerá contra ataques entre máquinas virtuais no mesmo servidor[32]. Nesse contexto é possível observar que as questões de segurança em nuvem são semelhantes as encontradas em *Data Center*.

Vale ressaltar que toda a responsabilidade da segurança e proteção da informação cabe as três partes envolvidas: o usuário da nuvem, o fornecedor da nuvem e quaisquer fornecedor terceiros [6].

2.2 Conceitos Básicos Sobre Dependabilidade

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 [9], ou ainda, que o sistema irá executar ações especificadas ou apresentar resultados específicos de maneira confiável e em tempo oportuno [60]. 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 [50]. 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 [67].

Em [8], é apresentada uma exposição sistemática dos conceitos de dependabilidade, que podem ser observados na Figura 2.2 e são mencionados a seguir:

- Os atributos: possibilitam a obtenção de medidas quantitativas, que muitas vezes

são cruciais para a análise dos serviços oferecidos.

- Os meios: são os meios pelos quais a dependabilidade é atingida.
- As ameaças: compreendem as falhas, erros e defeitos. A falha do sistema representa o evento que ocorre quando a entrega do serviço não acontece da maneira desejada.

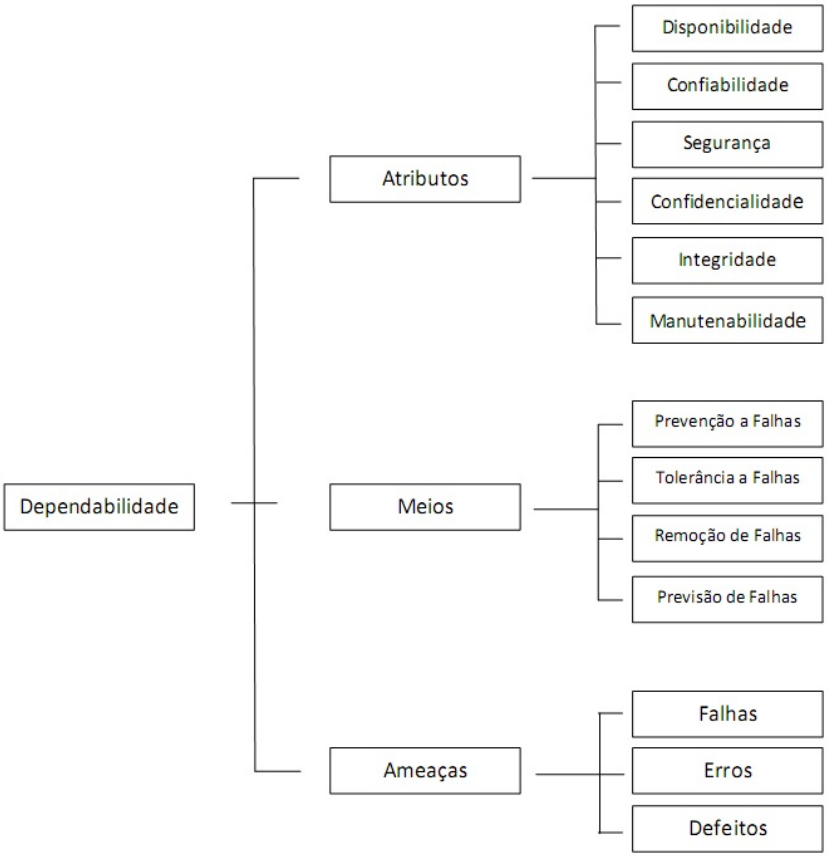


Figura 2.2: Árvore de dependabilidade
Adaptado de [8]

Falhas de *software* ou quebras de *hardware* em sistemas de nuvem pode afetar a disponibilidade e, assim, a comunicação entre seus componentes, tendo um impacto sobre a dependabilidade do sistema. Desse modo, técnicas como replicação de componentes específicos em determinada arquitetura em muitos sistemas de nuvem podem garantir os aspectos de dependabilidade apresentados e descritos a seguir.

2.2.1 Atributos de Dependabilidade

Os atributos de dependabilidade são: disponibilidade, confiabilidade, segurança, confidencialidade, integridade e manutenabilidade [8].

Confiabilidade

A confiabilidade é definida como a probabilidade de que um sistema irá executar a sua função pretendida durante um período de tempo de funcionamento sem qualquer falha [55]. Assim, a confiabilidade do sistema pode ser entendida como uma medida para a continuidade do serviço. Para descrever a confiabilidade de um dado sistema, é necessário conhecer a sua configuração, o estado em que ele é definido como operacional e as regras de operação [46].

Matematicamente, a função de confiabilidade $R(t)$ é a probabilidade de que o sistema irá funcionar corretamente, sem falha, no intervalo de tempo de 0 a t [46] [59], e pode ser vista através da Equação 2.1.

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

Onde T é uma variável aleatória, representando o tempo de falha ou 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.

Em ambientes em que se deseja ter alta disponibilidade, deve-se ter certeza de que equipamentos tenham passado por ambientes de testes durante certo período de tempo. Com isso, garantir que os equipamentos tenham passado pela primeira fase, ao mesmo tempo, deve-se tomar cuidado para que o equipamento seja substituído antes de entrar na fase final.

Por isso, 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 $MTTF$ (Equação 2.4) e $MTTR$ (Equação 2.5). 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 indica que o sistema se encontra funcionando durante 99,9876% do tempo e inativo em 0,0124% do tempo observado. O número de noves da disponibilidade pode ser calculada 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.

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

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

Assim, a análise da disponibilidade compreende o cálculo de índices como Tempo Médio para Falha (*Mean Time To Failure - MTTF*) e Tempo Médio para Reparo (*Mean Time To Repair - MTTR*), descritos a seguir:

- MTTF - é o tempo médio para a ocorrência de falhas no sistema.

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

- MTTR - é o tempo médio em que o sistema está indisponível devido a atividades de reparo. Os MTTRs estão relacionadas com a política de manutenção adotada pela organização e com as características inerentes a cada ação específica de reparo.

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

Onde UA representa a indisponibilidade do sistema, (Equação 2.6), e A a disponibilidade do sistema *(Availability)* (Equação 2.2).

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

Assim, o *downtime* anual (tempo de inatividade do sistema no período de um ano) pode ser calculada seguindo a Equação 2.7.

$$D = UA \times 8760h \tag{2.7}$$

A variação da taxa de falha dos componentes de *hardware* é mostrado na Figura 2.3. Conhecida como curva da banheira, a figura demonstra a taxa de falhas de componentes de hardware em três fases distintas [26].

Durante a primeira fase, 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, 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.

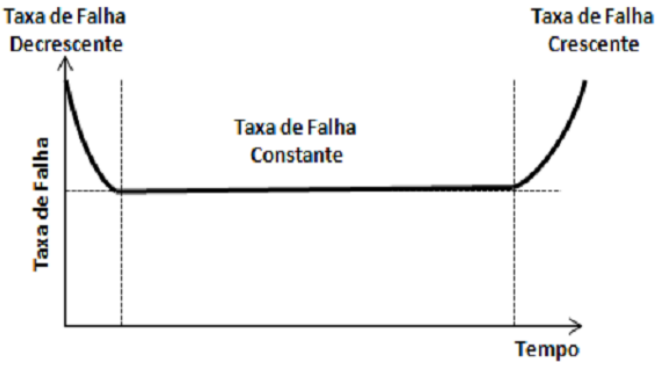


Figura 2.3: Curva da Banheira [26]

O estudo da disponibilidade do sistema pode ser dividida em três classes[63]: disponibilidade básica, alta disponibilidade e disponibilidade contínua.

um sistema de disponibilidade básica não requerem 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 2.4 demonstra 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.

Esse tipo de disponibilidade pode chegar, apenas, a três noves de disponibilidade.

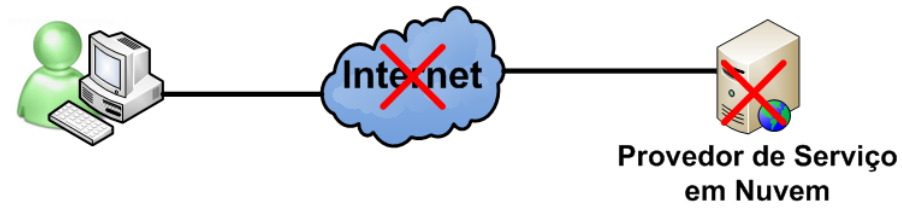


Figura 2.4: Exemplo de Sistema com Disponibilidade Básica

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.

Já 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.

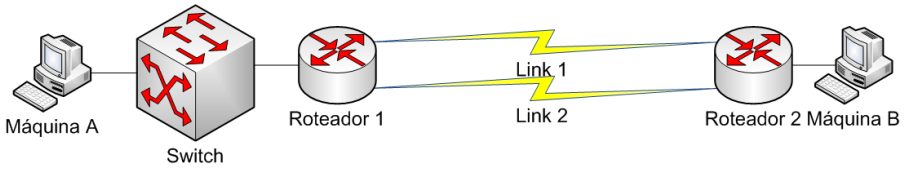


Figura 2.5: Exemplo de sistema de Alta Disponibilidade
Adaptado de [38]

A Figura 2.5 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 [38]. 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.

A disponibilidade contínua ou (*continuous availability*) visa manter um sistema totalmente disponível. Esse tipo de disponibilidade é muito mais ambiciosa em se comparando com os 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 2.6, 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

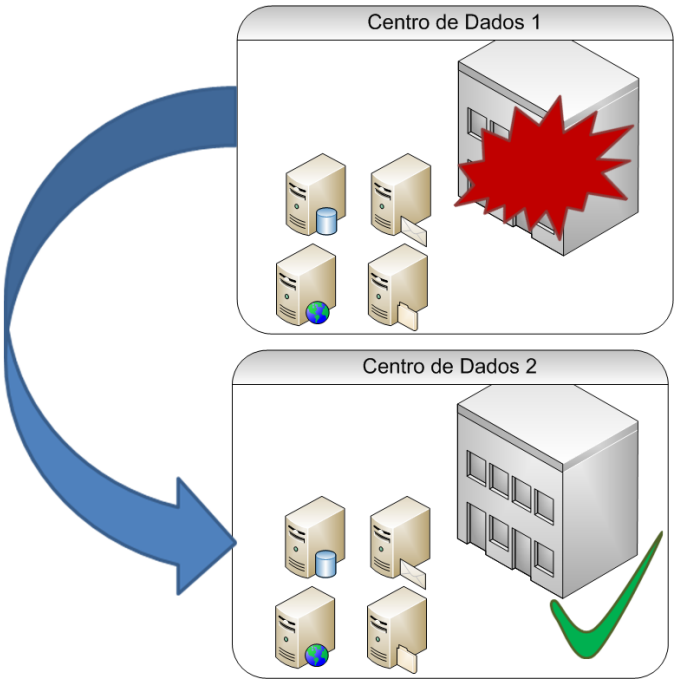


Figura 2.6: Exemplo de sistema de Disponibilidade Contínua

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, envolvendo toda uma infraestrutura de serviço.

Definindo técnicas e meios de alcançar a disponibilidade desejada, é possível calcular métricas para obtenção de resultados e tomadas de decisões adequadas. A disponibilidade orientada à capacidade é uma delas. Consideremos, por exemplo, um sistema com dois servidores paralelos, em que o sistema é considerado operacional se pelo menos um dos servidores estiver operacional. A disponibilidade orientada a capacidade (COA) [40], é uma métrica que indica não apenas se o sistema está operacional ou não, mas também a capacidade de atendimento que o sistema será capaz de oferecer.

Em um exemplo de um sistema com dois processadores. Sabemos que, o sistema poderá oferecer o serviço completo (os dois processadores em operação), metade do serviço (apenas um processador em execução) ou zero do serviço (todos os processadores em estado de quebra ou reparo). Assim, a disponibilidade orientada a capacidade no estado estacionário é dada por:

$$COA = \pi_2 + 0,5\pi_1 \tag{2.8}$$

De um modo mais simplificado, levando em consideração um sistema em que se tem uma disponibilidade orientada à capacidade igual à 99,9190%. Assim, dizemos que em 99,9190% de tempo de processamento está disponível ou são efetivamente entregues ao longo de um ano. Portanto, 0,0810% não está sendo entregue (indisponibilidade orientada a capacidade), ou seja, para um sistema com dois processadores, de 2*525.600 minutos de operação, 851 minutos o sistema não oferecerá seus serviços, ficando fora de operação.

Segurança

A segurança pode ser considerada uma extensão de confiabilidade [8]: a segurança está relacionado a confiabilidade no sentido de estar livre de danos catastróficos, não de perigo. Assim, pode-se dizer que a segurança em sistemas está diretamente ligada à ausência de consequências catastróficas.

Confidencialidade

A confidencialidade pode ser entendida como a ausência de divulgação não autorizada de informação [8]. As informações devem permanecer secretas e apenas pessoas autorizadas acessam as mesmas.

Acesso não autorizado a informações confidenciais podem ter consequências sérias, não só em aplicações de segurança nacional, mas também no comércio e na indústria [8].

Integridade

A integridade é um pré-requisito para a confiabilidade, disponibilidade e segurança [8]. Diferente da confidencialidade, a integridade tenta garantir que os dados não sejam alterados durante uma comunicação.

Mecanismos de proteção de integridade podem ser agrupados em duas grandes categorias [72]:

- Mecanismos de prevenção, tais como controles de acesso que impedem a modificação não autorizada de informação e;
- Mecanismos de detecção, que se destinam a detectar modificações não autorizadas quando os mecanismos de prevenção falharam.

Manutenabilidade

Quando um sistema falha ou deixa de oferecer os serviços, normalmente, aplicam-se operações de reparo para corrigir o problema. Assim, manutenabilidade é a capacidade de o sistema sofrer modificações e reparos [71]. A manutenabilidade pode ser descrita como a probabilidade de que um sistema seja reparado após a ocorrência de um evento de falha em um determinado período de tempo.

2.2.2 Meios de Dependabilidade

O desenvolvimento de um sistema de computação confiável requer a utilização combinada de um conjunto de quatro técnicas [8]:

- Prevenção a falhas: como prevenir a ocorrência de falhas,
- Tolerância a falhas: como oferecer um serviço correto na presença de falhas,
- Remoção de falha: como reduzir o número ou a gravidade de defeitos,
- Previsão de falhas: como estimar o número atual, a incidência futura, e as prováveis consequências de falhas.

Uma breve descrição relativas a essas técnicas são descritas nesta seção.

Prevenção a Falhas

A prevenção de falhas é alcançada por meio de técnicas de controle de qualidade utilizados durante o projeto e fabricação de hardware e software [8]. Partindo dessa perspectiva, prevenção de falha é a prevenção de ocorrência ou introdução de algum tipo de falha no sistema.

Tolerância a Falhas

A tolerância a falhas fornece o serviço esperado mesmo na presença de falhas. É geralmente implementada por detecção de erros e recuperação do sistema subsequente [8]. 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 [8]: verificação, diagnóstico e correção.

A **verificação** é o processo de verificar se o sistema adere propriedades dadas, ou seja, verificar se as funcionalidades do sistemas 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 conseqüências 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 [8]:

- **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

2.2.3 Ameaças à Dependabilidade

Um Servidor que está disponível por um longo período de tempo pode ter sua disponibilidade afetada de diversas maneiras, através de falhas, erros ou defeitos, porém esses três tipos de distorções têm significados distintos. As falhas estão associadas ao universo físico, erros ao universo da informação e defeitos ao universo do usuário [74]. Uma visão generalizada desses três tipos de problemas pode ser observada na Figura 2.7.

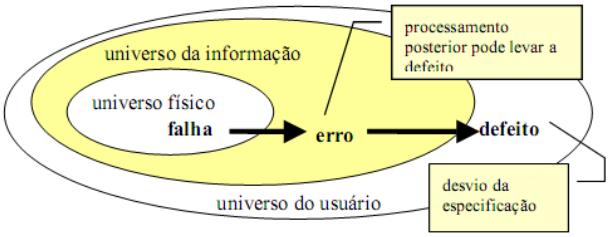


Figura 2.7: Modelo de 3 Universos: Falha, Erro e Defeito [74]

De uma forma mais fácil de entender podemos dizer, por exemplo, que a oscilação constante de energia pode ser considerada uma falha, e essa falha, por sua vez, pode gerar um erro no computador, na troca de bits, inesperadamente. Por conseqüência, seria gerado um defeito, podendo travar o equipamento ou até mesmo provocar a queima do material, afetando assim sua disponibilidade.

2.3 Técnicas para Avaliação 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 dependabilidades 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 modelação [51].

Eles podem ser classificados em modelos combinatórios e baseados em estado [50]. Modelos baseados em estado podem também se referir como não combinatórios, e modelos combinatoriais podem ser identificados como modelos não baseados em estados.

2.3.1 Cadeias de Markov

Uma Cadeia de Markov é um modelo baseado em estado no qual se diz que o estado atual não depende dos estados anteriores para que se conheçam os estados seguintes, tendo sido criado para a modelagem de sistemas. Assim, com esse formalismo é possível descrever o funcionamento de um sistema por meio de um conjunto de estados e transições.

As cadeias de Markov são modelos matemáticos úteis para a descrição de análises estatísticas que possuem valores de tempo em seus parâmetros. Assim, conhecidos como processo estocástico.

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 $(s_i \in S)$ [14]. 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 $X_0, X_1, X_2, ..., X_n, X_{n+1}$, $P(X_{n+1} = s_{n+1} | X_n = s_n, X_{n-1} = s_{n-1}, ..., X_0 = s_0) = P(X_{n+1} = s_{n+1} | X_n = s_n)$ [11].

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 do estado denominado [50]. 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 [22]. 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 [5].

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 [5].

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 2.10.

$$Q = \begin{pmatrix} q_{ii} & q_{ij} \\ q_{ji} & q_{jj} \end{pmatrix}, \tag{2.9}$$

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

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, $||\pi|| = 1$ [5]. 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 2.11.

$$p_{i,j}(s,t) = PX(t) = j|X(s) = i \tag{2.11}$$

A solução transiente, ou dependente do tempo, é importante quando o sistema a avaliar é dependente do tempo [22]. Para modelos ergódicos, considerando tempos de execução longos, pode-se mostrar que a probabilidade dos estados converge para valores constantes [41]. 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, $\pi_i = \lim_{t\to\infty} \pi_i(t)$(homogeneidade), consequentemente, $\pi'(t) = 0$, resultando nas Equações 2.10 e 2.12.

$$\sum_{i=1}^{N} \pi_i = 1 \tag{2.12}$$

A Equação 2.12 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. Por isso, modelos de Markov por recompensa estão sendo usados e a sua descrição formal pode ser vista a seguir.

Cadeias de Markov com Recompensa

Definição 1. *Um modelo de Markov por recompensa (*Markov Reward Model - MRM*) M é uma 3-tupla ((S,**R**,Rótulo),ρ,ι)) em que (S,**R**,Rótulo) é uma CTMC subjacente rotulado C, $\rho : S \to \Re_{\geq 0}$ é a estrutura de recompensa de estados, e $\iota : S \times S \to \Re_{\geq 0}$ é a estrutura de recompensa por impulso de tal modo que se $\boldsymbol{R}_{S,S} \geq 0$ então $\iota(s,s) = 0$.*

Uma MRM é um CTMC rotulada e aumentada com recompensa por estado e estruturas de recompensa por impulso. A estrutura de recompensa por estado é uma função ρ que

atribui a cada estado $s \in S$ uma recompensa de $\rho(s)$ tal que se t unidades de tempo são gastos no estado s, uma recompensa de $\rho(s) \cdot t$ é adquirido. As recompensas que são definidas na estrutura de recompensa por estados podem ser interpretadas de várias formas. Elas podem ser consideradas como o ganho ou benefício adquirido por ficar em determinado estado e também podem ser consideradas como o custo incorrido por ficar em algum estado.

A estrutura de recompensa de impulso, por outro lado, é uma função ι que atribui a cada transição de s para s', onde $s, s' \in S$, uma recompensa $\iota(s, s')$ de tal modo que, se a transição de s para s' ocorre, uma recompensa de $\iota(s, s')$ é adquirida. Semelhante à estrutura de recompensa por estado, a estrutura de recompensa de impulso pode ser interpretada de várias formas. Uma recompensa de impulso pode ser considerada como o custo de tomar uma transição ou o ganho que é adquirida, levando em consideração a transição.

2.3.2 Diagramas de Bloco de Confiabilidade

Diagrama de bloco de confiabilidade (RBD) foi a primeira técnica apresentada para determinar a confiabilidade de um sistema. Em um modelo de diagrama de blocos, os componentes são representados como blocos e são combinados com outros blocos, ou seja, combinados com outros componentes, em série, paralelo, ou configurações *k-out-of-n* [69]. Assim, RBD's possibilitam a modelagem de sistema, fornecendo uma visualização gráfica dos componentes do sistema. O modelo RBD fornece uma iteração lógica entre componentes do sistema, definindo quais combinações ativas definem a funcionalidade do sistema. Com essa técnica, é possível também calcular as métricas de dependabilidade, tais como a disponibilidade e manutenabilidade.

A Figura 2.8 mostra dois exemplos de modelos RBD's, onde os blocos são organizados em série (Figura 4.2) e paralelo (Figura 4.5). No modelo em série, se um único componente falhar, o sistema para de prover o serviço entrando no modo de falha.

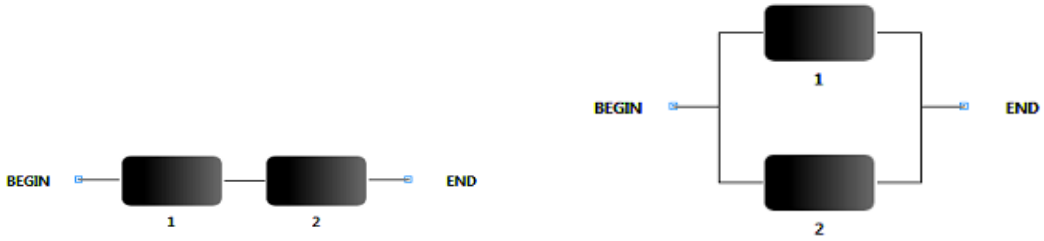
A confiabilidade de dois blocos conectados em série (ver Figura 4.2) pode ser obtida através da Equação 2.13.

$$R_s(t) = R_1(t) \times R_2(t) \tag{2.13}$$

Onde:

$R_1(t)$ é a confiabilidade do bloco 1.

$R_2(t)$ é a confiabilidade do bloco 2.



(a) Série (b) Paralelo

Figura 2.8: Diagrama de Blocos de Confiabilidade

De uma forma geral, considerando um sistema com n elementos, a confiabilidade do modelo em série pode ser obtida através da Equação 2.14.

$$R_s(t) = \prod_{i=1}^{n} R_i(t) \tag{2.14}$$

Onde:

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

A confiabilidade de dois blocos conectados em paralelo (ver Figura 4.5) pode ser obtida através da Equação 2.15.

$$R_p(t) = 1 - \prod_{i=1}^{2} (1 - R_i(t)) \tag{2.15}$$

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 Equação 2.16.

$$R_p(t) = 1 - \prod_{i=1}^{n} (1 - R_i(t)) \tag{2.16}$$

onde:

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

Modelos RBD's 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.

2.3.3 Métodos para Alta Disponibilidade em *Cluster*

Em sistemas que necessitam de alta disponibilidade e confiabilidade, é necessário criar métodos para a detecção e correção de falhas, evitando defeitos do sistema. Uma falha de grande escala em um sistema pode significar perdas catastróficas. Muitas técnicas têm sido propostas e adotadas para construir *clusters* de *failover* [49] bem como a virtualização e influência de sistemas em nuvem, para abordar as questões de confiabilidade de serviço [76] [15].

Muitas técnicas baseiam-se em redundância, i.e, a replicação de componentes de modo que trabalhem para um fim comum, na tentativa de garantir a segurança de dados e a disponibilidade do serviço, mesmo em caso de falhas de componentes. Porém, vários outros métodos e técnicas têm sido adotados para este fim.

Redundância para Alta Disponibilidade em *Cluster*

Sistemas com requisitos rigorosos de confiabilidade exigem métodos para detectar, corrigir, evitar e tolerar falhas e defeitos. Uma falha em um sistema de grande escala pode significar perdas catastróficas. Muitas técnicas têm sido propostas e aprovadas para construir *clusters* de *failover* [49] bem como virtualização e sistemas de nuvem para abordar questões de serviços de dependabilidade [76] [15]. Muitas dessas técnicas são baseadas em redundância, isto é, a replicação dos componentes, de modo que eles funcionem para uma finalidade comum, garantindo a segurança e a disponibilidade dos dados, mesmo no caso de alguma falha de componente. Três técnicas de replicação merecem atenção especial devido à sua ampla utilização em infraestruturas de servidores em *cluster* [16] [50]:

Cold Standby: Essa técnica consiste em ter um módulo de *backup* desligado em espera e só será ativado se o nó primário falhar. O ponto positivo para esta técnica é que o nó secundário tem baixo consumo de energia e não usa o sistema. Por outro lado, o nó secundário precisa de um tempo significativo para ser ativado e clientes que estavam acessando informações sobre o nó primário perdem todas as informações com a falha do nó primário. Muitas vezes o usuário terá que refazer boa parte dos trabalhos quando o nó secundário for ativado.

Hot Standby: Esta técnica pode ser considerada a mais transparente dos modos de replicação. Os módulos replicados são sincronizados com o módulo operacional, assim, os participantes ativos e *clusters* em *standby* são vistos pelo usuário final como um único recurso. Depois de um nó falhar, o nó secundário é ativado automaticamente e os usuários que acessam o nó principal agora acessam o nó secundário sem notar a mudança de equipamento.

Warm Standby: Esta técnica tenta equilibrar o custo e o atraso do tempo de recuperação das técnicas de espera *Cold* e *Hot Standby*. O nó secundário está em modo de espera, mas não completamente desligado, para que possa ser ativado o mais rápido possível do que na técnica *Cold Standby*, assim que um monitor detectar a falha do nó principal. O nó replicado é parcialmente sincronizado com o nó operacional, de modo que, se o usuário adicionar informações no nó operacional primário, não perderá informações que estavam sendo escritas no momento em que o nó principal falhou.

A replicação de dados e atividades de monitoração de serviços ou recursos é a mais importante em um sistema desse nível. Para construir *clusters* de *failover*, compostas por dois ou mais nós, existem técnicas de mascaramento de falhas que devem ser adotadas. Em sistemas baseados em Linux, essas duas atividades são normalmente realizados pelo *heartbeat* [39] e DRBD (*Distributed Replicated Block Device*) [25].

Figura 2.9: *Heartbeat* e *DRBD*, *cluster* de dois nós

O Heartbeat é um *software* de gerenciamento de *cluster* que permite, numa infraestrutura de *cluster*, identificar os *hosts*, ativos e inativos, por meio de troca de mensagens periódicas. A Figura 2.9 mostra um *cluster* com dois nós, na tentativa de manter um determinado serviço de forma ininterrupta. O *Heartbeat* iria enviar mensagens de um nó para outro nó, detectando quando o *host1* entrar em modo *offline*, ativando assim os serviços no *host2*, sem interrupções perceptíveis ao usuário final.

O dispositivo de bloco distribuído e replicado (DRBD) é um módulo para o *kernel* do Linux que fornece um *driver* de dispositivo de bloco [62]. Em cada nó do *cluster*, o *driver DRBD* está no controle de um dispositivo de bloco "real", que detém uma réplica de dados do sistema. Operações de leitura são realizadas localmente, enquanto que operações de escrita são transmitidas para os outros nós do *cluster* de alta disponibilidade.

O DRBD trabalha como RAID1[1] em rede, assim ele é responsável por sincronizar os dados entre os *hosts*. O espelhamento de dados do DRBD usa um dos três protocolos [62].

[1]Duplicação de 100% dos dados em dois discos, (para saber mais seguir a referência [18])

- **Protocolo A** - é completamente assíncrona. Aplicações são notificadas de conclusão de escrita quando as gravações tenham concluído localmente, o que geralmente acontece antes que eles tenham se propagado para outros *hosts*.
- **Protocolo B** - pares de nós mais estreitamente do que o anterior. A Conclusão de uma operação de gravação é assinalada para as camadas superiores do sistema operativo, assim como o local de IO é completo e um pacote de reconhecimento chega a partir do nó secundário. Este reconhecimento é enviado pelo nó secundário logo que receba a operação de gravação.
- **Protocolo C** - é completamente síncrona. Aplicações são notificados da conclusão de escrita após a escrita ter sido realizada em todos os *hosts*.

Cada dispositivo DRBD pode estar no estado primário ou secundário. Aplicações são concedidas com acesso de gravação apenas se o dispositivo estiver em estado primário. Apenas um dispositivo de um par de dispositivos ligados pode estar no estado primário. A atribuição dos papéis (primário ou secundário) para os dispositivos geralmente é feita por softwares de gerenciamento de *cluster*, como o *Heartbeat*.

2.4 Considerações Finais

Este capítulo apresentou os conceitos básicos sobre computação nas nuvens. Em seguida, conceitos básicos sobre avaliação de dependabilidade foram apresentados. Então, técnicas para avaliação de dependabilidade foram descritas. Finalmente, técnicas para garantir a alta disponibilidade de sistemas foram apresentadas.

3

Infraestrutura de Computação em Nuvem Eucalyptus

Seu trabalho irá tomar uma grande parte da sua vida
e o único meio de ficar satisfeito é fazer o que você
acredita ser um grande trabalho.

—STEVE JOBS

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 [28]. 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* [30]. A compatibilidade da *API* permite executar um aplicativo da *Amazon* no *Eucalyptus* sem modificação. O *Eucalyptus* foi desenvolvido na Universidade da Califórnia, Santa Barbara, com o propósito de pesquisa em computação em nuvem.

Em geral, a plataforma do *Eucalyptus* utiliza a capacidade de virtualização (*hypervisor*) do sistema de computador subjacente para permitir alocação flexível de recursos de computação dissociados de *hardware* específico [28]. *Eucalyptus* é compatível com pacotes para múltiplas distribuições do *Linux*, incluindo o *Ubuntu, RHEL, OpenSuse, Debian, Fedora*, e o *CentOS*.

Há cinco componentes de alto nível da arquitetura *Eucalyptus*, cada um com seu próprio *web service*: *Cloud Controller, Cluster Controller, Node Controller, Storage Controller*, e *Walrus* [28]. A Figura 3.1 mostra um exemplo de um ambiente base de computação em nuvem *Eucalyptus*, considerando dois *clusters* (A e B). Uma breve

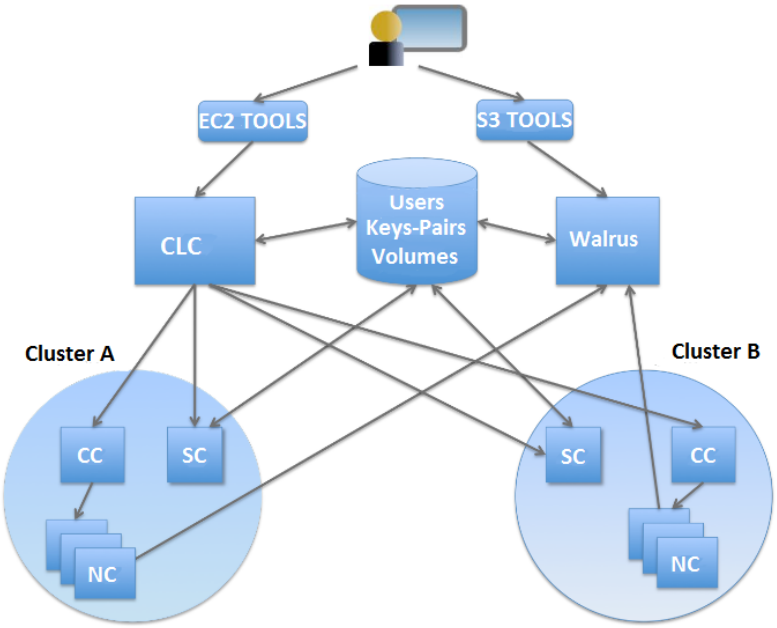


Figura 3.1: Componentes de alto nível *Eucalyptus* [29]

descrição dos componentes do sistema *Eucalyptus* pode ser vista a seguir:

3.1 Controlador da Nuvem (CLC)

O Controlador da Nuvem (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 [68]. 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 [20]:

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

3.2 Controlador do *Cluster* (CC)

O Controlador do *Cluster* (CC) geralmente executa em um *cluster* de máquinas *front-end* [28] [27] ou em qualquer máquina que tenha conectividade na rede para ambos os nós que executam os NCs e para a máquina que se encontra executando o CLC. CCs reúnem informações sobre um conjunto de máquinas virtuais e horários de execução de VM no NCs específico. O CC tem três funções principais: agendar visitas, implantação e execução para NCs específico; controlar a instância virtual de rede *overlay* e reunir/relatar informações sobre um conjunto de NCs [27]. 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 suas funções são divididas da seguinte forma [20]:

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

3.3 Controlador do Nó (NC)

O Controlador do Nó (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 [28]. 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 [27], [20]. Sendo assim, podemos dividir as funções do NC da seguinte forma [20]:

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

3.4 Controlador de Armazenamento (SC)

O Controlador de armazenamento (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 [27], 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 [20], [1]. A sua função é dividida desse modo [20]:

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

3.5 *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) [27]. 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 [27], [20]. 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 [20]:

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

3.6 Arquitetura de Nuvem Privada

A Figura 3.2 mostra uma arquitetura genérica simples composto por um *front-end* e seus Nós.

Um computador *front-end* é adotado como o "Subsistema de *Cloud*"e configurado com os componentes do *Eucalyptus* conhecidos e mostrados anteriormente, controlador da nuvem (CLC), *walrus*, controlador do *cluster* (CC) e controlador de armazenamento (SC). Para essa arquitetura simples, existe três controladores de nós (NC) configurados e executando, essas máquinas são responsável por iniciar e gerenciar as máquinas virtuais, interagindo diretamente com o *hypervisor*. A interação entre clientes e a Nuvem é realizada por meio de uma interface *web* facilitando a comunicação e interação entre usuários e o sistema.

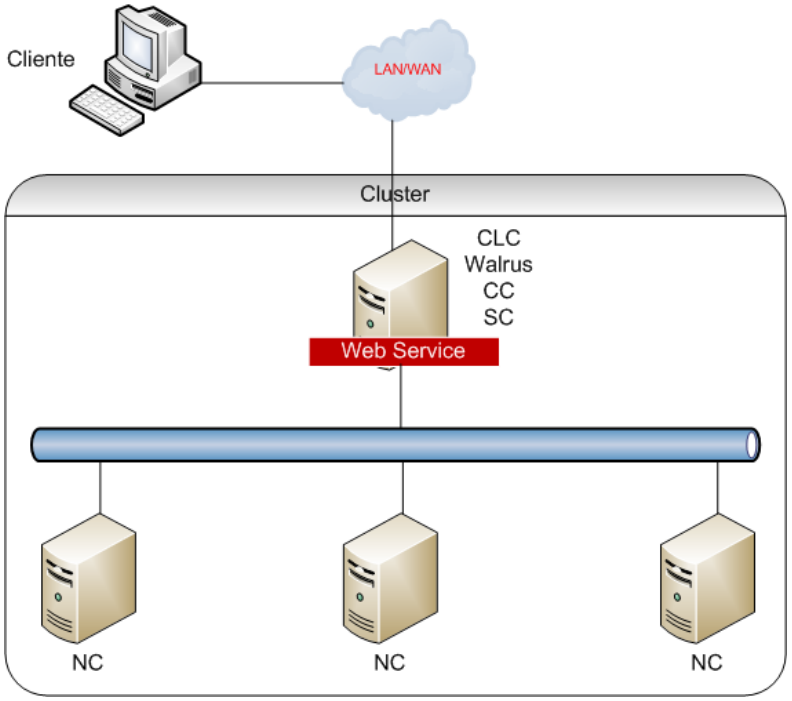


Figura 3.2: Modelo de uma Arquitetura Genérica Simples *Eucalyptus*

Em sua instalação o *Eucalyptus* pode agregar e gerenciar recursos de um ou vários *clusters*. Para a construção de um *cluster* consiste pelo menos de duas máquinas: uma rodando o CC, SC, *walrus* e o CLC, e a outra, rodando o NC. Recomendada para ambientes de experimentação. Uma instalação em múltiplos clusters pode colocar cada um de seus componentes em máquinas separadas. É preferido quando se tem a intenção de realizar trabalhos mais sérios.

3.7 Adições para Eucalyptus

Euca2ools - o *euca2ools* é uma ferramenta de linha de comando do Eucalyptus. Estes comandos ajudam a gerenciar imagens, instâncias, armazenamento, rede, etc., através de uma interação com *web services*, sendo compatível com os serviços EC2 e S3 da Amazom.

ElasticFox - O *ElasticFox* é uma extensão aberta do *Mozilla Firefox* para interagir com a *Amazon Compute Cloud* (Amazon EC2). É usado para lançar novas instâncias, montagem de volumes EBS, mapa elástico de endereços IPs e muito mais. O *ElasticFox* foi originalmente escrito para trabalhar com o Amazon EC2, mas, desde a versão 1.7, foi adicionado como suporte para o *Eucalyptus*, justamente por ter APIs compatíveis.

Hybridfox - Corresponde a uma modificação do *ElasticFox* para torná-lo utilizável com o *Eucalyptus*. Enquanto que o *ElasticFox* trabalha apenas com a AWS, o *Hybridfox* pode ser usado como uma interface única para a AWS e o *Eucalyptus*. A interface do *Hybridfox* é semelhante à do *ElasticFox*. Mesmo após o recente lançamento de *ElasticFox* (versão 1.7) com suporte para *Eucalyptus*, o *Hybridfox* tem as seguintes características adicionais que o tornam atraente para os usuários do *Eucalyptus*: aumentar a quantidade de instâncias com o IP Privado; algumas peculiaridades relacionadas ao mapeamento de código rígido de tipos de instância e tipos de arquitetura abordadas; suporte para *Eucalyptus* 1.5.x bem como 1.6.x; outras melhorias de usabilidade [20].

3.8 Considerações Finais

Este capítulo apresentou os conceitos básicos sobre a ferramenta *Eucalyptus*, sendo expostos os módulos do sistema separadamente, exemplificando suas principais caracteristicas e funções. Posteriormente, foi mostrada um exemplo de uma arquitetura de nuvem *Eucalyptus* e em seguida algumas adições para a ferramenta *Eucalyptus*, facilitando o entendimento e o funcionamento do sistema.

4

Metodologia e Modelos

A única maneira de fazer um excelente
trabalho é amar o que você faz.
—STEVE JOBS

O desenvolvimento de técnicas, estratégias e modelos que proporcionem meios para avaliação de dependabilidade de sistemas de computação em nuvem é de fundamental importância para empresas que proveem serviços através dessa infrestrutura, pois *hardware* e *software* falham, contudo os usuários não devem ser privados dos serviços.

Este capítulo apresenta a metodologia para realização deste trabalho, os modelos RBDs, que possuem diferentes níveis de abstração, e o modelo CTMC capaz de representar arquiteturas redundantes e demonstrar, através de formulas fechadas, o cálculo da disponibilidade do sistema.

4.1 Metodologia

A metodologia utilizada para avaliar a dependabilidade da infraestrutura de sistemas em nuvem compreende a realização de cinco etapas, são elas: conhecimento e levantamento de informações do sistema *Eucalyptus*, definição dos componentes relevantes do sistema, a serem avaliados, geração dos modelos dos subsistemas, construção do modelo do sistema, avaliação e interpretação dos resultados. Neste trabalho, o processo de modelagem adota uma estratégia hierárquica que utiliza modelos combinatoriais e modelos baseados em estado para representar as características de dependabilidade do sistema.

- **Conhecimento e levantamento de informações do sistema:** Essa fase refere-se à compreensão do sistema, seus componentes, suas interfaces, interações e funcionamento. Nessa fase, é importante a identificação do problema a ser analisado;

Figura 4.1: Metodologia Proposta

além disso, devem-se definir os requisitos de dependabilidade a serem alcançados, que serão considerados no processo de avaliação.

- **Definição dos componentes do sistema:** Os componentes a serem utilizados para avaliação do sistema, bem como falhas de *hardware* e *software* a serem consideradas para avaliação de disponibilidade. É nessa fase que agrupamos os componentes que trabalham em conjunto de modo a gerar os primeiros modelos do subsistema para a próxima fase.

- **Geração dos modelos do subsistema:** A partir dos componentes coletados na fase anterior, são gerados pequenos modelos de subsistema do sistema *Eucalyptus*. A partir desses subsistemas, é possível extrair as primeiras métricas de tempo de falha e de reparo para serem utilizadas na próxima fase.

- **Construção do modelo do sistema:** Nessa fase é gerado o sistema completo, a partir dos subsistemas gerados na fase anterior.

- **Avaliação:** Com o modelo do sistema completo, é possível obter resultados de

disponibilidade, custo com construção de determinadas arquiteturas bem como capacidade de processamento das mesmas. Assim, os resultados são apresentados.

As avaliações dos modelos de subsistemas em RBDs visam encontrar parâmetros que sirvam de entrada para o modelo CTMC. O modelo CTMC representa o comportamento de um subsistema redundante.

4.2 Modelos

Esta seção apresenta os modelos RBDs e CTMC's. Baseados nos estudos de características e componentes descritos no capítulo anterior, alguns modelos analíticos foram criados para descrever o comportamento de componentes dos sistemas. São consideradas falhas de *hardware* e *software*. Portanto, os componentes utilizados para representar um computador são: *hardware*, sistema operacional e os componentes do *Eucalyptus* (a especificação de cada componente a ser utilizado depende do nível da arquitetura na nuvem). São gerados modelos RBDs dos componentes do sistema individualmente, além do modelo CTMC para representar redundância.

4.2.1 Modelo RBD para Componentes Individuais

O primeiro modelo concebido representa os componentes que são integrados para formar o computador que representa o ponto de entrada de clientes na nuvem (computador principal da arquitetura). Para este computador, são gerados dois modelos diferentes: o primeiro chamado de controlador geral ou *general controller* (GC), que comporta grande parte dos componentes envolvidos, sendo eles: *hardware* (HW), sistema operacional (SO), *cloud controller* (CLC), *cluster controller* (CC), *storage controller* (SC) e *walrus*. A Figura 4.2 mostra o modelo usado para representar o (GC).

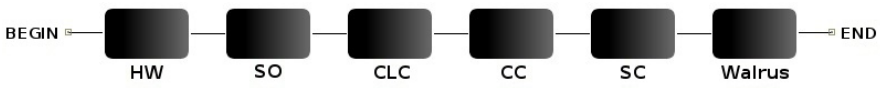


Figura 4.2: Modelo RBD do Subsistema do Controlador Geral

O segundo modelo apresenta os componentes *Eucalyptus* subdivididos. Esse segundo modelo foi assim dividido a fim de separar o controlador da nuvem do controlador do *cluster*. O modelo é chamado de Subsistema de Nuvem. Esse nível do sistema consiste em *hardware* (HW), sistema operacional (so) e alguns componentes de *software Eucalyptus*, *cloud controller* (CLC) e *walrus*, mostrados na Figura 4.3.

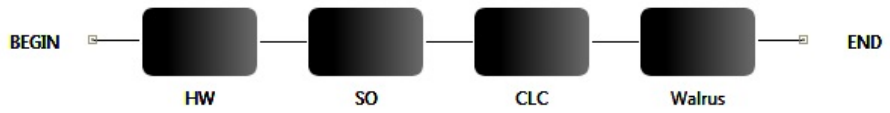


Figura 4.3: Modelo RBD do Subsistema de Nuvem

A Figura 4.4 apresenta um modelo RBD do subsistema de *cluster*, considerando-se os valores dos MTTFs e MTTRs para os sistemas de *cluster* individuais. Para este componente, foi utilizado o *hardware* (HW), sistema operacional (so), *cluster controller* (CC) e *storage controller* (SC).

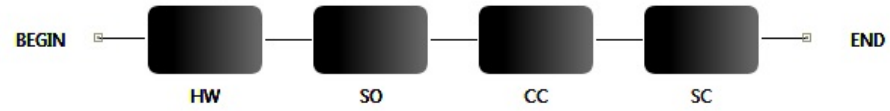


Figura 4.4: Modelo RBD do Subsistema de *Cluster*

Para os modelos da Figura 4.2, 4.3 e 4.4, são utilizados valores como parâmetros de entrada encontrados na Tabela 4.1, os componentes do sistema *Eucalyptus* são, em sua maioria, baseados em composições *web-services*, assim sendo, os valores utilizados como base foram retirados de [43] e para os demais componentes foram retirados de [45].

Tabela 4.1: Parâmetros de Entrada para o Controlador Geral [43][45]

Componente	**MTTF**	**MTTR**
HW	8760 h	100 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

Por fim, é apresentado o modelo concebido para representar o Subsistema de Nós, cada arquitetura da nuvem é composto por um ou mais Nós no sistema. Para cada componente de nó é considerado que possuam os componentes, além dos já apresentados, *hardware* (HW) e sistema operacional (so). Esse modelo possui também um modulo *KVM* e o componente do *Eucalyptus node controller* (NC). O modelo que representa esse subsistema pode ser visto na Figura 4.5.

Para este modelo, assume-se que *hardware* e sistema operacional possuem as mesmas características dos modelos apresentados anteriormente, tendo os mesmos valores

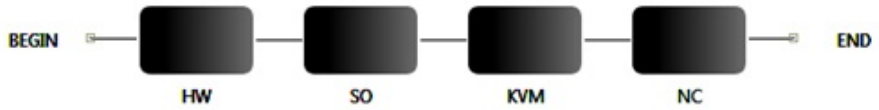


Figura 4.5: Modelo RBD do Subsistema de Nós

apresentados na Tabela 4.1. Assim, os valores do MTTF usados para o *KVM* e *node controller* foram retirados de [45] [43] e podem ser observados na Tabela 4.2.

Tabela 4.2: Parâmetros de Entrada para o Subsistema de Nós [43][45]

Componente	MTTF	MTTR
KVM	2990 h	1 h
NC	788.4 h	1 h

4.2.2 Modelo CTMC para Avaliação da COA

Para avaliação da COA, foi criado um modelo CTMC mostrado na Figura 4.6. O modelo retrata uma MRM onde são atribuídas recompensas para cada estado da cadeia, representando o número de núcleos de processamento disponíveis para toda a arquitetura. O modelo descreve o Subsistema de *cluster* e o Subsistema de nó descrito na seção anterior. Este modelo possui 8 estados: CC_F1, CC_F2, CC_F3, CC_F4, estados esses que representam a quebra do *cluster*, e os estados $6K$, $4K$, $2K$ e $0K$ representando a quantidade de nós ativos no *cluster* através da quantidade de núcleos disponíveis e de nós ativos.

Os estados que representam os nós, $6K$, $4K$, $2K$ e $0K$, recebem uma recompensa indicando o número de núcleos de processamento disponível. No estado $6K$, o sistema está completo, ou seja, com três nós habilitados fornecendo 100% do número de núcleos disponíveis e com o subsistema de *cluster* ativo. Com a falha do subsistema de *cluster*, o sistema pode entrar nos estados CC_F1, CC_F2 ou CC_F3 (estados pintados em cinza) representando a falha do *cluster*, parando de fornecer os serviços. Se um nó falhar, o sistema vai para o estado $4K$, representando a falha de um nó. Se outro nó entrar em estado de falha, antes da recuperação do nó anterior, o sistema vai para o estado $2K$, onde temos um nó habilitado e o subsistema de *cluster* também ativado. O estado $0K$ representa a falha dos três nós. Neste estado, o sistema para de fornecer os serviços. Na ocorrência da falha de um nó com o sistema de *cluster* no estado de falha (CC_F1) o sistema passa diretamente para o estado CC_F2, representando que o sistema continua

em estado de falha porém sem o funcionamento de um de seus nós. É importante salientar que com a falha de um dos nós no sistema e a falha do *cluster* representado pelo estado CC_F2, há uma prioridade de reparo sobre o *cluster* possibilitando que o sistema volte a fornecer os serviços mesmo com um nó em falha.

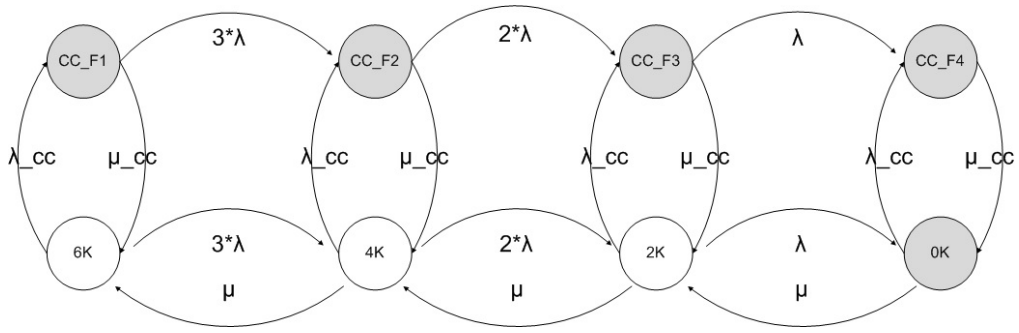


Figura 4.6: Modelo CTMC para o Sistema com um Subsistema de *Cluster*

A taxa de λ_cc denota a taxa de falha do subsistema de *cluster*. A taxa de reparo, tanto CC_F1, CC_F2, CC_F3 e CC_F4, é de μ_cc. A taxa de $3 \times \lambda$ representa a taxa de falha do estado $6k$, sendo atribuída para representar o peso no estado quando se tem os três nós ativos (computadores no mesmo *cluster*). A Tabela 4.3 apresenta os valores de todos os parâmetros mencionados da MRM.

Tabela 4.3: Parâmetros de Entrada para o Modelo CTMC

Parâmetros	**Descrição**	**Valores**
$1/\lambda\_cc$	Tempo Médio para Falha do *Cluster*	$333,71$ h
$1/\lambda$	Tempo Médio para Falha do Nó	$481,83$ h
$1/\mu$	Tempo Médio de Reparo do Nó	$0,91$ h
$1/\mu\_cc$	Tempo Médio de Reparo do *Cluster*	$0,94$ h

Os valores para as taxas λ_cc e μ_cc foram obtidos a partir do MTTF e MTTR computados usando o modelo RBD do subsistema de *Cluster* (Figura 4.4) e λ e μ foram retirados do modelo RBD do subsistema de nós (Figura 4.5).

$$COA_{System} = \frac{\sum_{i=1}^{n}(COA_{cluster_i})}{n}, \tag{4.1}$$

A COA de cada arquitetura analisada pode ser computada usando a Equação 4.1.

Onde $(COA_{cluster_i})$ representa a COA de cada n *clusters* composta na arquitetura e é computado com uma recompensa por estado correspondente no modelo MRM (ver

Figura 4.6), podendo ser expressa pela Equação 4.2.

$$COA_{cluster_i} = \sum_{s \in S} \pi_s \times \rho_s \tag{4.2}$$

onde π_s é a probabilidade de estado estacionário de estar em um determinado estado s da MRM. ρ_s é a taxa de recompensa atribuído ao estado s e é equivalente a fração do número total de núcleos disponíveis naquele estado. Para as arquiteturas analisadas nesse estudo de caso, as taxas de recompensa de r_i são 1, $\frac{2}{3}$, and $\frac{1}{3}$ para os estados 6K, 4K, e 2K respectivamente. A taxa de recompensa para todos os outros estados é 0 porque o *cluster* está em falha, assim não há núcleos de processador disponíveis. Com isso é possível também obter uma equação de forma fechada para computar a disponibilidade de todo o sistema de nuvem (A_{system}), a partir dos modelos RBD correspondentes. A Equação 4.3 denota como computar a disponibilidade do sistema, de acordo com a regra de composição de componentes série e paralelo do modelo RBD [46]. Esta é uma equação geral para uma arquitetura composta de k *clusters*. A_{CLC} é a disponibilidade do bloco representando o Subsistema de Nuvem. Ao lidar com um sistema Subsistema de Nuvem redundante o valor de A_{CLC} será computada com a Equação 4.4. $A_{cluster_j}$ é a disponibilidade de cada Subsistema de *Cluster* calculado usando o modelo MRM da Figura 4.6.

$$A_{System} = (A_{CLC}) \times (1 - \prod_{j=1}^{k}((1 - A_{cluster\_j}))) \tag{4.3}$$

4.2.3 Modelo CTMC para Arquitetura redundante

A fim de aumentar a disponibilidade das infraestruturas, propõe-se a aplicação de redundância nos componentes críticos do sistema. Mecanismos de *software*, tais como DRBD e o *Heartbeat*, descrito no Capítulo 2, podem ser utilizados para este fim, uma vez que eles reforçam a coerência dos dados e a disponibilidade do serviço. Estratégias de replicação não podem ser adequadamente representadas em modelos RBD, devido à dificuldade de representar redundância entre componentes do sistema. Assim, o subsistema do controlador geral redundante é representado por um modelo de Markov por Recompensa (MRM), mostrado na Figura 4.7, o qual teve sua definição formal descrita no Capítulo 2.

O modelo MRM na Figura 4.7 tem cinco estados: UW, UF, FF, FU, e FW. A recompensa atribuída aos estados UW, UF e FU são iguais a 1, uma vez que o subsistema

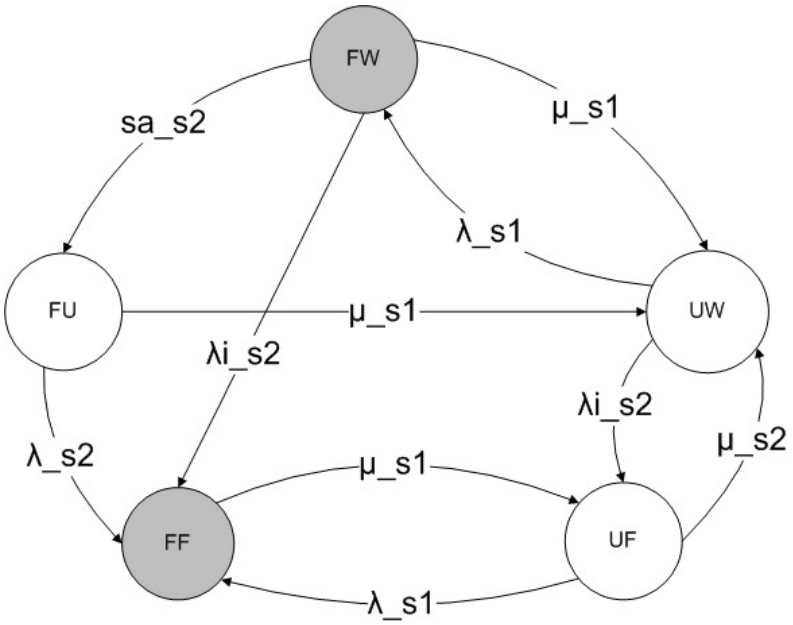


Figura 4.7: Modelo CTMC para o Sistema Redundante com Dois GC

do controlador geral (GC) está disponível nesses estados. A recompensa atribuída aos estados FF e FW (estados sombreados) são iguais a 0, uma vez que o subsistema de GC é considerado inativo ou em quebra nesses estados. Não existem recompensas por impulso neste modelo, recompensas associadas a transições entre os estados. Por isso, a disponibilidade de estado estacionário do subsistema GC pode ser calculada como a recompensa de estado estacionário do MRM. Assim, $A_{GC} = \sum_{s \in S} \pi_s \cdot \rho(s)$, onde π_s é a probabilidade de estado estacionário de estar no estado s, e $\rho(s)$ é a recompensa atribuída ao estado s.

No estado UW, o GC1 principal está ativo e o GC2 secundário está em uma condição de espera. Quando o GC1 falhar, o sistema entra no estado FW, onde o GC secundário ainda não detectou a falha de GC1. Consideramos ainda a possibilidade de falha do servidor secundário antes do tempo de ativação do mesmo, atribuindo uma taxa λ_i_s2 representando a falha do sistema secundário, dessa forma o sistema pode sair do estado FW (estado que representa a falha do GC1 e o GC2 em espera) e ir para o estado FF (representando a falha dos dois servidores), a taxa de falha do GC secundário em espera é 20% menor do que o índice de falha de um GC ativo, já que o mesmo encontra-se ligado, mas não operacional. FU representa o estado onde GC2 sai da condição de espera e assume o papel de controlador geral ativo, enquanto GC1 está em estado de falha. Se GC2 falhar antes da reparação de GC1, o sistema vai para o estado FF. A fim de dar prioridade à reparação do servidor principal, há apenas uma única passagem de reparação

de FF, que vai para UF. Se o GC2 falhar quando GC1 está ativo, o sistema vai para o estado UF, retornando ao estado UW com o reparo de GC2, ou vai para o estado FF em caso de GC1 também falhar. As taxas de falha de GC1 e GC2 são indicadas por λ_s1 e λ_s2, respectivamente. A taxa de λ_i_s2 denota a taxa de falha do servidor secundário quando ele está inativo. A taxa de reparo de GC2 é de μ_s2. A taxa de transição sa_s2 representa a taxa de passagem, ou seja, o inverso do tempo médio para ativar o servidor secundário após uma falha de GC1. A Tabela 4.4 apresenta os valores de todos os parâmetros mencionados da MRM. O valor de μ_s1 é igual ao valor de μ_s2, as taxas de λ_s1 e λ_s2 também possuem valores iguais. Os valores de λ e μ foram obtidos a partir do MTTF e MTTR retirados da avaliação usando o modelo único RBD para o Controlador Geral (ver Figura 4.2). O valor de sa_s2 é retirado dos arquivos de configuração do *software Hearbeat*, como mencionado. O *software* necessita de um intervalo padrão de monitoramento para detectar a falha dos componentes configurados para desempenhar determinadas funções. Por padrão, assume-se que o sistema irá ativar o controlador secundário no intervalo de tempo de 0,005 ($\cong$ 18 segundos).

Tabela 4.4: Parâmetros de Entrada para o Modelo CTMC

Parâmetros	**Descrição**	**Valores**
$\lambda\_s1 = \lambda\_s2 = 1/\lambda$	Tempo Médio de Falha do *host*	180,72 h
$\lambda_i\_s2 = 1/\lambda_i$	Tempo Médio para Falha do *host* inativo	216,86 h
$\mu\_s1 = \mu\_s2 = 1/\mu$	Tempo Médio de Reparo do *host*	0,97 h
$sa\_s2 = 1/sa$	Tempo Médio para Ativar o Sistema	0,005 h

O modelo da Figura 4.7 permite a obtenção de uma expressão algébrica para a disponibilidade do controlador do subsistema redundante geral, como pode ser visto na Equação 4.4.

$$A_{GC} = \frac{\mu(\lambda \times \lambda_i + (\lambda_i + \mu)^2 + sa(\lambda + \lambda_i + \mu))}{sa(\lambda^2 + \lambda(\lambda_i + \mu) + \mu(\lambda_i + \mu)) + (\lambda + \mu)(\lambda \times \lambda_i + (\lambda_i + \mu)^2)} \tag{4.4}$$

É também possível a obtenção de uma expressão algébrica para calcular a disponibilidade da infraestrutura da nuvem completa (A_{cloud}), através do modelo RBD mostrado na Figura 5.2, com base nos estudos vistos nos capítulos anteriores. A Equação 4.5 calcula a disponibilidade de acordo com a regra de composição de componentes em série e paralelo [46].

$$A_{cloud} = A_{GC} \times (1 - \prod_{i=1}^{n}(1 - A_{Node\_i})) \tag{4.5}$$

4.3 Considerações Finais

Este capítulo apresentou a metodologia de avaliação adotada neste trabalho. Em seguida, os modelos RBDs para componentes individuais foram apresentados para, só então, apresentar os modelos RBDs das arquiteturas com dois e três níveis. Também foram apresentados e conceituados os modelos CTMCs concebidos. Para cada um dos modelos RBDs para componentes individuais apresentados, foram extraídas as falhas (MTTFs) do sistema que servem como parâmetros de entrada para o modelo CTMC. Para cada modelo CTMC, foram apresentadas expressões algébricas para cálculo da disponibilidade estacionária e orientada à capacidade.

5
Estudo de Caso

Já fiz todos os cálculos; o destino fará o resto.

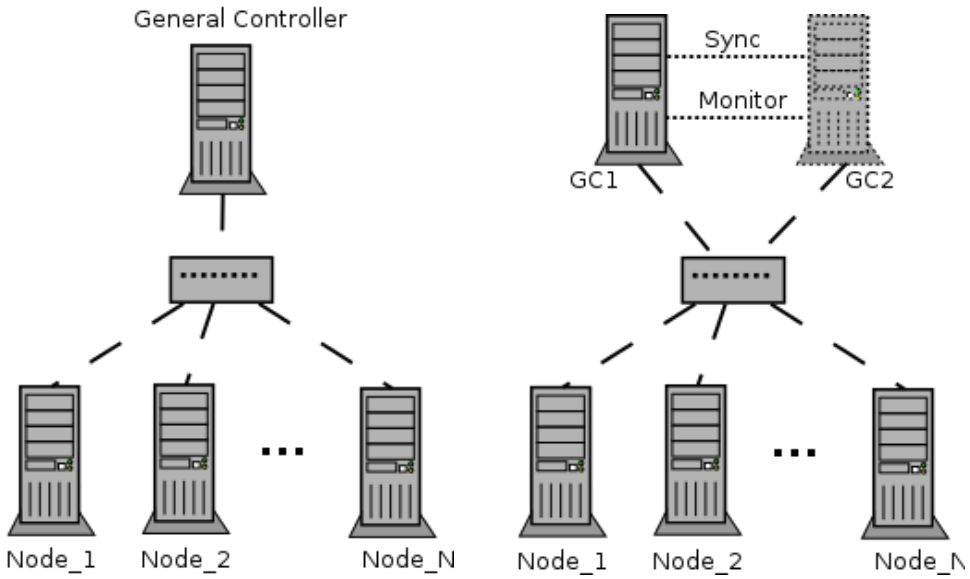
—NAPOLEÃO BONAPARTE

Este capítulo apresenta um cenário para avaliação do impacto da disponibilidade em ambientes de computação em nuvem para o sistema *Eucalyptus*. Posteriormente, um conjunto de cenários é apresentado para avaliação do impacto da disponibilidade em sistema da nuvem, sendo combinados a fim de demonstrar melhores cenários para construção de sistemas da nuvem, avaliando métricas como disponibilidade no estado estacionário, disponibilidade orientada a capacidade e custo de construção dos cenários apresentados individualmente. Os resultados das avaliações permitem identificar melhores arquiteturas para construção de sistemas em nuvem.

5.1 Primeiro Estudo de Caso

Baseado nos estudos e características descritas no Capítulo 3, o primeiro estudo de caso consiste em verificar os efeitos de um mecanismo de replicação em um dos componente do sistema. O estudo de caso analisa a disponibilidade de duas arquiteturas de nuvem privada, sendo ambos os componentes de *hardware* e *software* considerados.

A Figura 5.1a retrata a primeira arquitetura, onde todos os principais componentes do *Eucalyptus* - Controlador da Nuvem, Controlador do *Cluster*, Controlador de Armazenamento e *Walrus* - estão em execução em uma mesma máquina, chamada de controlador geral (GC). Há N nós disponíveis para implantar VMs, por meio do Controlador de Nó executando em cada máquina individual (N_1, N_2, ..., N_N). Esta arquitetura simples possui um único ponto de falha no controlador geral, de modo que qualquer falha de *hardware* ou *software* pode ocasionar na falta da entrega do serviço.



(a) Não Redundante (b) Redundante no GC

Figura 5.1: Arquitetura de Nuvem Privada

5.1.1 Modelo para Arquitetura não Redundante

Devido à sua simplicidade e eficiência de computação, diagramas de blocos de confiabilidade (RBDs) são usados para analisar a disponibilidade dos sistemas e, agrupados a ele, um Modelo de Markov por Recompensa (MRM) é usado para representar a segunda arquitetura redundante. O RBD descreve um modelo de dois níveis, enquanto que a MRM representa os componentes envolvidos no mecanismo de redundância.

Este estudo de caso considera um exemplo de uma arquitetura com cinco nós, onde pelo menos um nó deve estar disponível para a nuvem funcionar corretamente. Portanto, o subsistema de Nó é representado por um conjunto de cinco blocos de nós em paralelo. A Figura 5.2 mostra o modelo RBD completo com um Controlador Geral (não redundante) e seus cinco nós. Os valores utilizados para o bloco que representa o Controlador Geral (GC) não redundante e o Nó foram retirados dos modelos 4.2 e 4.5 descritos no Capítulo 4.

A disponibilidade desta arquitetura não redundante foi calculada usando este modelo RBD e é mostrada na Tabela 5.1, com as suas medidas relacionadas: o número de noves [52], que constitui uma vista logarítmica da disponibilidade; e tempo de inatividade, a que melhor representa o impacto da indisponibilidade dos serviços quando se tem clientes acessando determinados serviços. Uma vez que o *downtime* anual chega a 46,66 horas, é de salientar que a disponibilidade do sistema não é tão alta o quanto se espera de uma

infraestrutura em nuvem.

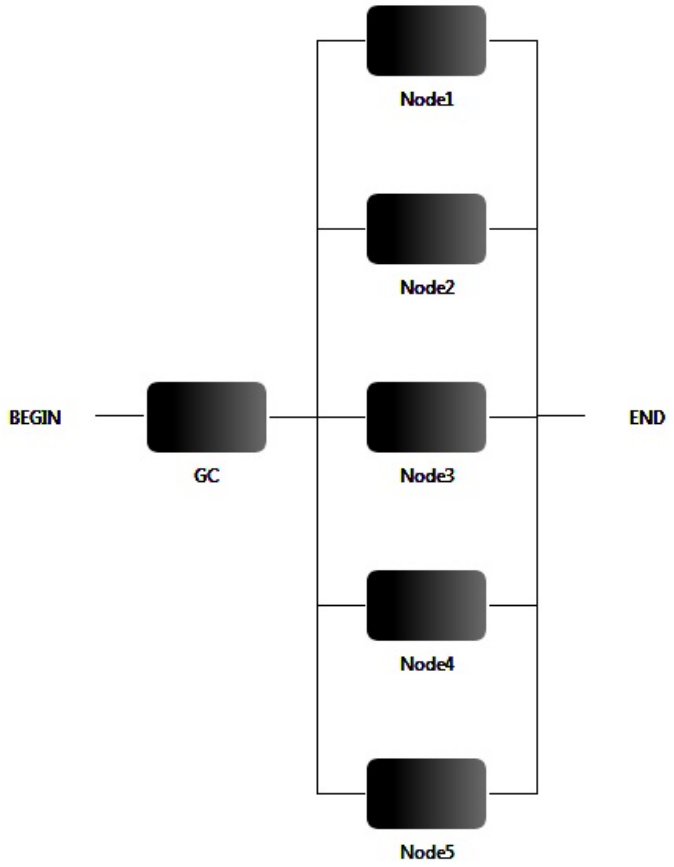


Figura 5.2: Modelo RBD do Sistema de Nuvem não Redundante

5.1.2 Modelo para Arquitetura Redundante

A segunda arquitetura, descrita na Figura 5.1b, implementa redundância no controlador geral (GC2), seguindo uma estratégia *warm-standby*. Todos os componentes de *software* do primeiro controlador geral (GC1) são também instalados na máquina em espera (GC2), sendo ativado apenas quando uma falha no GC1 é detectada. Monitoramento contínuo e mecanismos de sincronização de dados permitem a transição rápida do serviço para o *host* secundário, caracterizando a abordagem da redundância *warm-standy*.

A fim de aumentar a disponibilidade da infraestrutura mencionada na seção anterior, propõe-se a aplicação de um mecanismo de redundância no controlador geral, que é o único ponto de entrada para os clientes na nuvem. O modelo usado para representar a arquitetura redundante foi apresentado na Figura 4.7.

5.1.3 Resultados

Os resultados são mostrados na Tabela 5.1. São apresentados para a arquitetura não redundante (GC não Redundante) e para a arquitetura redundante (GC Redundante) a disponibilidade no estado estacionário, o número de noves e o *downtime* anual.

Tabela 5.1: Medidas de Disponibilidade do Sistema com/sem Redundância

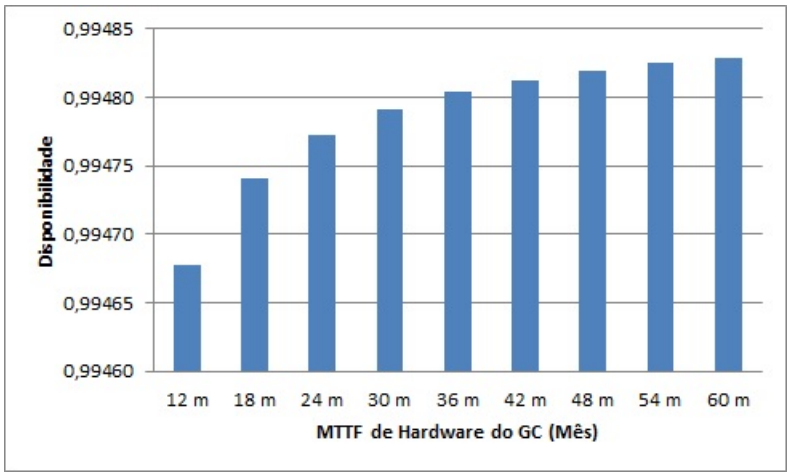
Medidas	**GC não Redundante**	**GC Redundante**
Disponibilidade	0,99467823178	0,99991779
Números de 9's	2,273944	4,08510
Downtime Anual	46,66 h	0,72 h

Estes resultados, que comparam o sistema com e sem redundância, revelam um aumento na disponibilidade do sistema, que corresponde a uma redução de 98,45% do *downtime* anual. É importante salientar que os benefícios da replicação não podem ser alcançados simplesmente aumentando a confiabilidade do *hardware* no GC, isto é, através da aquisição de uma máquina com um maior tempo médio de falha. Este fato é demonstrado pela análise de comparação representada nas Figuras 5.3a e 5.3b.

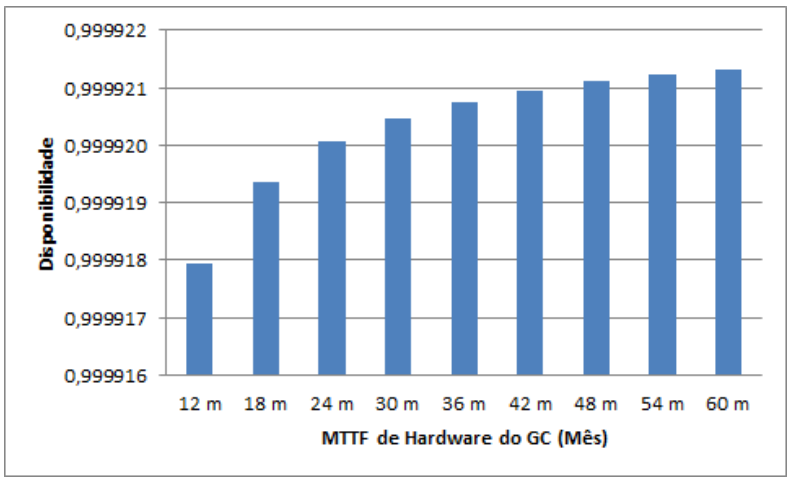
Mesmo se o MTTF do *hardware* no GC for alterada para 60 meses, a disponibilidade do sistema não redundante não chegaria à disponibilidade do sistema redundante com *hardware* menos confiável. Portanto, a implementação de um mecanismo *warm-standby* de replicação no GC é uma estratégia valiosa para alcançar alta disponibilidade em ambientes em nuvem com base no sistema *Eucalyptus*.

5.2 Segundo Estudo de Caso

Este estudo de caso analisa várias arquiteturas possíveis para a construção de um sistema de nuvem Eucalyptus redundante, comparando a disponibilidade e os custos de cada opção. Os computadores que compõem cada arquitetura analisada têm características idênticas: mesmo fabricante, modelo, configuração, e custos [23], apresentado na Tabela 5.2. A Figura 5.4 mostra uma arquitetura genérica simples, que tem componentes não redundantes de software *Eucalyptus* e é composta por três *clusters*. Um computador *front-end* é adotado como o "Subsistema de Nuvem", configurado com os componentes do *Eucalyptus* conhecidos como Controlador da Nuvem (CLC) e *Walrus*. Cada *cluster* é um computador chamado de "Subsistema de *Cluster*", que executa o controlador do *cluster* (CC) e o controlador de armazenamento (SC). Cada *cluster* também tem três máquinas



(a) Arquitetura não Redundante



(b) Arquitetura Redundante

Figura 5.3: Disponibilidade do Sistema vs MTTF de *Hardware* do Controlador Geral

que executam os componentes do *Eucalyptus* conhecidos como controladores de nó, responsáveis por instanciar e gerenciar as máquinas virtuais, interagindo diretamente com o *hypervisor*. O conjunto de três nós em cada *cluster* é chamado de "Subsistema de Nó". Além da arquitetura, com três *clusters*, sistemas com um e dois *clusters* também são avaliados neste estudo de caso. O impacto da implementação da redundância no subsistema de Nuvem (CLC e *Walrus*) e no subsistema de *Cluster* (CC e SC) é considerado para esses sistemas. Todas as arquiteturas replicadas seguem uma estratégia de replicação *warm-standby*, que pode ser implementada através dos *softwares* como DRBD [25, 62] e *Heartbeat* [39].

Tabela 5.2: Descrição do Computador

Marca/Modelo	Componentes	Descrição
DELL/Power Edge	HD	1 TB
	Memória	8 GB
	CPU	Intel Xeon - 2.2GHZ
Custo Total (USD)		1339,00

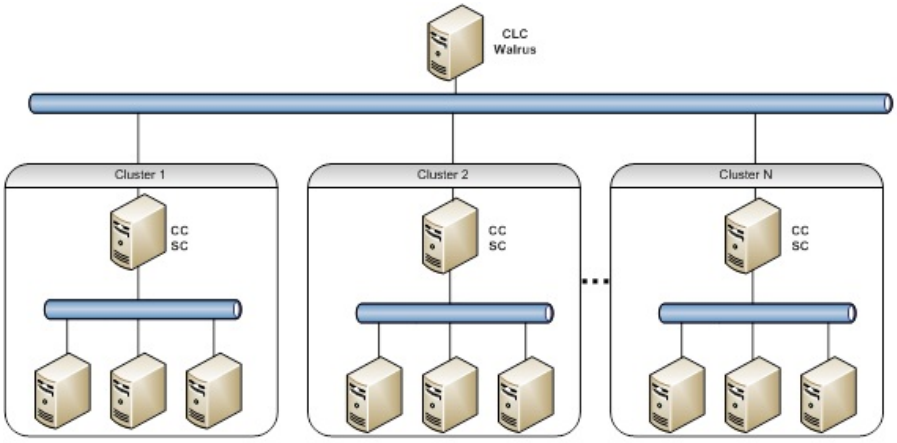


Figura 5.4: Arquitetura de Nuvem Privada

5.2.1 Arquiteturas de Nuvem Privada Redundante

O modelo RBD para essa arquitetura é subdividida em três níveis e traz alguns dos componentes *Eucalyptus* do controlador geral separados. Portanto, essa arquitetura pode ser dividida em três partes: Subsistema de nuvem, Subsistema do *cluster* e o Subsistema de nós. Esses três subsistemas já foram apresentados no capítulo anterior.

Os modelos RBD's foram criados para avaliar três cenários, com três níveis divididos em subsistemas. Para cada cenário, considera-se que possuem números diferentes de *clusters* e nós . O Cenário I consiste em um subsistema de nuvem, um Subsistema de *Cluster* e três *hosts* no Subsistema de nós (ver Figura 5.5).

A Figura 5.6 representa o cenário II e é composta de um Subsistema de nuvem, dois subsistemas de *clusters* e seis *hosts* organizados igualmente em dois subsistemas de nós.

A Figura 5.7 representa o cenário III e é composto por um Subsistema de nuvem, três subsistemas de *Clusters* e nove *hosts* divididos em três subsistemas nós.

Em todos os cenários, o sistema está disponível se, e somente se, o subsistema de Nuvem estiver funcionando e executando, ou se pelo menos um Subsistema de *cluster* estiver disponível, com um ou mais nós em execução em determinado *cluster* ativo. Os modelos gerados são concebidos com a finalidade de obter resultados de disponibilidade

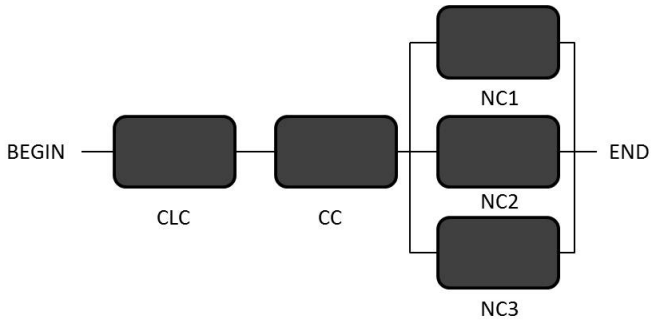


Figura 5.5: Modelo RBD para o Cenário I: Sistema de Nuvem com um *Cluster*

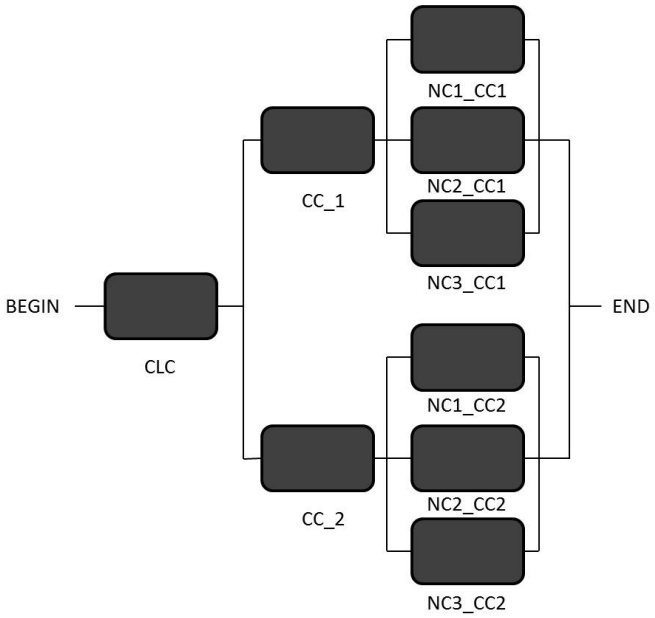


Figura 5.6: Modelo RBD para o Cenário II: Sistema de Nuvem com Dois *Clusters*

para os cenários propostos. Os resultados podem ser vistos no tópico a seguir.

5.2.2 Resultados

Para cada cenário (um, dois e três *clusters*) mencionado no Capítulo 4, três variantes redundantes foram avaliadas: (1) a redundância apenas no Subsistema da Nuvem, (2) a redundância apenas no Subsistema de *cluster* e (3) a redundância em ambos os Subsistemas, de nuvem e de *cluster*. As nove arquiteturas resultantes são enumeradas na Tabela 5.3. As Arquiteturas $A1$, $A2$ e $A3$ consideram apenas a redundância no subsistema da nuvem, com um, dois e três *clusters*, respectivamente. As Arquiteturas $A4$, $A5$, $A6$ consideram apenas a redundância no subsistema *Cluster*. As Arquiteturas $A7$, $A8$, $A9$

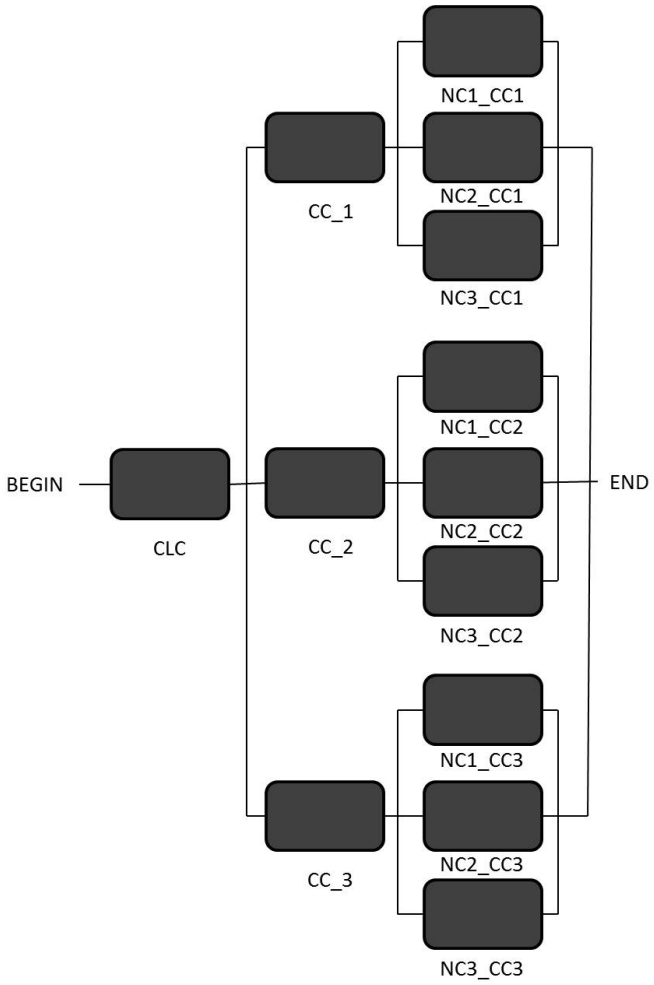


Figura 5.7: Modelo RBD para o Cenário III: Sistema de Nuvem com Três *Clusters*

consideram a implementação de redundância no subsistema de nuvem e *cluster*. Para todas as arquiteturas, o modelo correspondente RBD foi composto com o modelo MRM apresentado na Figura 4.7.

A Tabela 5.4 mostra os resultados deste estudo, considerando a disponibilidade em estado estacionário, o número de noves [52], o tempo de inatividade anual e o custo total para cada arquitetura. As arquiteturas com maiores índices de disponibilidade são as Arquiteturas $A8$ e $A9$. Elas correspondem aos cenários com dois e três *clusters*, empregando redundância tanto no subsistema do *CLC* quanto no subsistema do *CC*. Este resultado indica que o número de *clusters* não tem impacto significativo sobre a disponibilidade quando os componentes principais do *Eucalyptus* (*CLC* e *CC*) são instalados de modo a serem componentes redundantes no sistema. Devido a essa semelhança nos resultados de disponibilidade, a escolha entre as Arquiteturas $A8$ e $A9$ é baseada nos custos, os quais

Tabela 5.3: Descrição das Arquiteturas

Arquitetura	**Número de *clusters***	**Descrição**
A1	1	Redundância apenas
A2	2	para o Subsistema de
A3	3	Nuvem
A4	1	Redundância apenas
A5	2	para os Subsistemas de
A6	3	*Cluster*
A7	1	Redundância para o
A8	2	Subsistema de Nuvem e
A9	3	*Cluster*

são mais baixos para $A8$. Também é importante destacar a diferença de cerca de 24 horas no *downtime* vivido pela arquitetura pior ($A1$) e os melhores ($A8$ e $A9$). Apesar do menor custo de aquisição de $A1$, o tempo disponível pode incorrer em custos, tais como perdas de receita e penalidades devido à violação de SLA (*Service Level Agreement*).

Tabela 5.4: Medidas de Disponibilidade e custo do sistema, com e sem Redundância

Arquitetura	**Disponibilidade(%)**	**N. de 9's**	**Downtime (h)**	**Custo (U$)**
A1	99,716771	2,54786	24,81 h	8.034,00
A2	99,996533	4,46042	0,30 h	13.390,00
A3	99,997318	4,57200	0,23 h	18.746,00
A4	99,716771	2,54786	24,81 h	8.034,00
A5	99,719443	2,55197	24,58 h	14.729,00
A6	99,719443	2,55197	24,58 h	21.424,00
A7	99,994640	4,27131	0,47 h	9.373,00
A8	99,997320	4,57238	0,23 h	16.068,00
A9	99,997320	4,57239	0,23 h	22.763,00

A Figura 5.8 permite a comparação da indisponibilidade ($100 - Disponibilidade_i$) e custo, resultando em uma visão unificada. Os valores, da indisponibilidade foram normalizados através da Equação 5.1, onde, I_i representa a indisponibilidade da Arquitetura i e I_{max} o maior valor de indisponibilidade encontrado dentre as arquiteturas propostas, deixando assim, o valor de IN_i (Indisponibilidade Normal) em função de 0 (zero) a um.

$$IN_i = I_i \div I_{max} \tag{5.1}$$

Da mesma forma, os valores do custo de aquisição de cada arquitetura foram normalizados através da Equação 5.2, onde, C_i representa o custo da Arquitetura i e C_{max} o maior valor do custo encontrado dentre as arquiteturas propostas, deixando assim o valor de

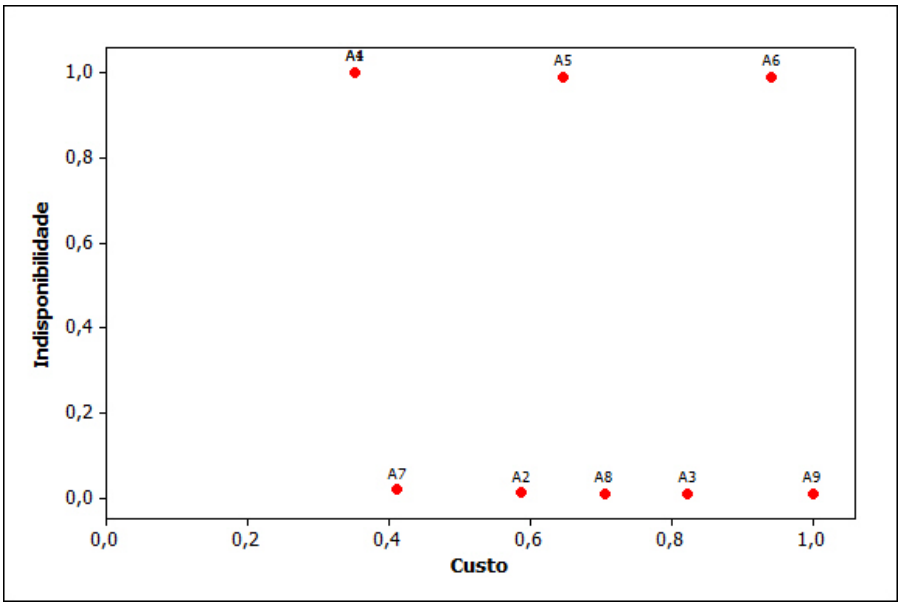


Figura 5.8: Resultados por Arquitetura: Comparando Indisponibilidade e Custo

CN_i (Custo Normal) em função de 0 (zero) a um.

$$CN_i = C_i \div C_{max} \tag{5.2}$$

Os valores, mostrados no gráfico, que se aproximam da origem demonstram os melhores resultados. A Tabela 5.5 justifica o resultado da melhor arquitetura por meio da distância euclidiana, distância do resultado de cada arquitetura até a origem (ponto zero), podendo ser calculado através da Equação 5.3, assim, a menor distância dentre as arquiteturas propostas é a Arquitetura A7, justificando assim nossos resultados, proporcionando baixo índice de indisponibilidade com um custo relativamente baixo, em se comparando com as demais.

A diferença do *downtime* anual de $A7$ para $A8$ e $A9$ é apenas cerca de 14 minutos (0,24 horas), reforçando como adequada é a Arquitetura $A7$.

$$D_i = \sqrt{(IN_i)^2 + (CN_i)^2} \tag{5.3}$$

É possível observar que as Arquiteturas $A5$ e $A6$ são más escolhas, porque o emprego de redundância não abaixa os índices de indisponibilidade em um grau elevado, quando comparado com $A2$ e $A3$, por exemplo, apesar dos custos de tais ações. A existência de um ponto único de falha (não redundante no Subsistema de Nuvem) explica os maus resultados das Arquiteturas $A5$ e $A6$.

Tabela 5.5: Distância Euclidiana dos Resultados para cada Arquitetura

Arquitetura	Distância
A1	1,06
A2	0,59
A3	0,82
A4	1,06
A5	1,18
A6	1,37
A7	0,41
A8	0,71
A9	1,00

Se apenas as arquiteturas com maior capacidade computacional ($A3, A6 e A9$), ou seja, aquelas com três *clusters*, fossem consideradas, a melhor opção seria $A3$, porque tem uma boa disponibilidade em um nível semelhante ao $A6$ e $A9$, e menor custo. Este é um resultado importante, pois indica que, para ambientes de nuvem privada com três ou mais *clusters*, há pouco benefício em empregar uma série redundante para o Subsistema de *cluster*. A redundância na série do Subsistema de Nuvem é suficiente para atingir um elevado grau de disponibilidade do sistema.

5.3 Terceiro Estudo de Caso

Com as arquiteturas mencionadas no estudo anterior, pretende-se, agora, comparar a disponibilidade no estado estacionário, analisando a disponibilidade orientada à capacidade. A Figura 5.4, mostrada no estudo anterior, demonstra a arquitetura a ser analisada. A mesma é subdividida em três partes: Subsistema de Nuvem, Subsistema de *cluster* e Subsistema de Nó. Para este estudo, a arquitetura fica, assim, melhor dividida para análise dos resultados propostos (disponibilidade x COA). Para análise da COA, é considerado que o Subsistema da Nuvem não impacta nos resultados. Por isso, é dispensado o uso desse subsistema para tal efeito. Considerando apenas o Subsistema de *cluster*. A Figura 5.9 demonstra o modelo a ser considerado para análise da COA, é considerado que para cada nó tenha um conjunto de dois núcleos totalizado seis núcleos por Subsistema de *cluster*. Assim, o impacto da implementação da redundância é considerado, apenas, para o subsistema de Nuvem.

Neste estudo, levam-se em conta que para cada Subsistema de *cluster* tenha disponível três nós, assim, é gerado um conjunto com cinco cenários com números distintos de *clusters*. O cenário I (Figura 5.10) consiste do Subsistema de Nuvem e um Subsistema

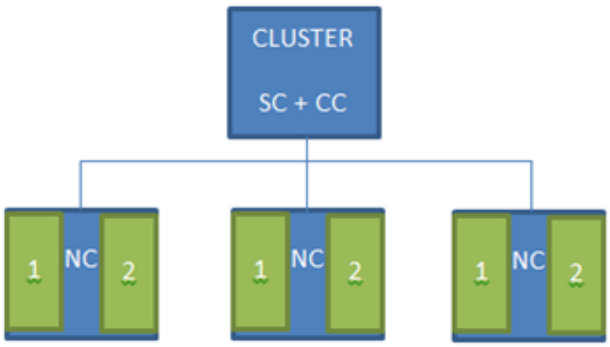


Figura 5.9: Quantidades de Núcleos por Nó

de *Cluster*. O cenário II (Figura 5.11) é composto do Subsistema de Nuvem e dois Subsistema de *Cluster*. O cenário III (Figura 5.12) consiste em um Subsistema de Nuvem e três Subsistemas de *Cluster*. O cenário IV (Figura 5.13) consiste em um Subsistema de Nuvem e quatro Subsistemas de *Cluster*. O cenário V (Figura 5.14) consiste em um Subsistema de Nuvem e cinco Subsistemas de *Cluster*. Para todos os cenários, o sistema está disponível se o Subsistema de Nuvem estiver disponível e se, pelo menos, um Subsistema de *Cluster* estiver disponível com no mínimo um *host* funcionando no Subsistema de Nó do *cluster*.

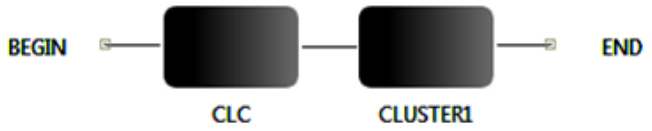


Figura 5.10: Cenário I

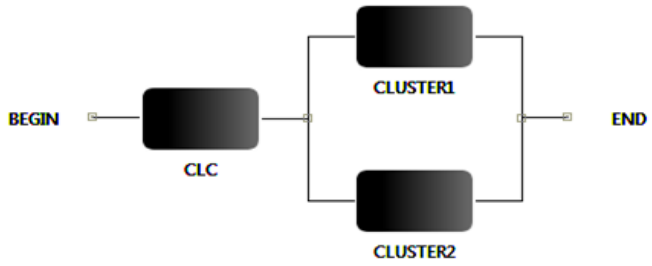


Figura 5.11: Cenário II

É importante salientar que o modelo CTMC descrito no Capitulo 4 e mostrado na Figura 4.6 representa o Subsistema de *cluster* (ver Figura 5.9), os resultados deste

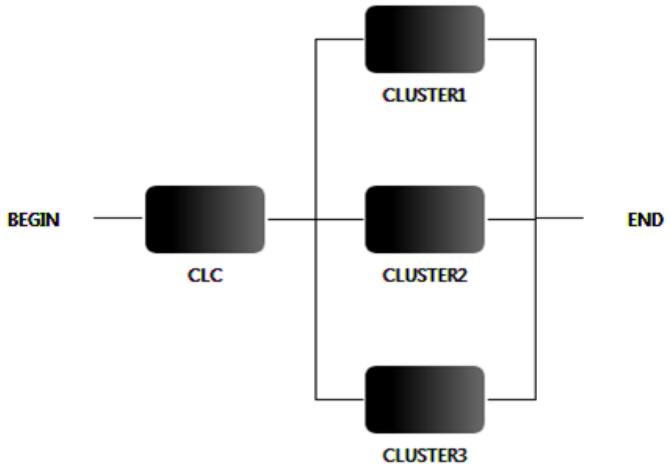


Figura 5.12: Cenário III

modelo são usados como parâmetros de entrada para os blocos RBD's que representam o Subsistema de *cluster* dos modelos apresentados anteriormente.

5.3.1 Resultados

É considerada para o cálculo da COA a atribuição de taxas de recompensa para cada estado do modelo de disponibilidade dos nós (sendo assim que esta cadeia de Markov torna-se modelo de Markov por recompensa). As recompensas atribuídas para cada estado representam o número de núcleos disponíveis por computador. A partir daí pode ser calculada a capacidade de processamento para cada arquitetura. Assim, as taxas de recompensa de r_i são 1, $\frac{2}{3}$, and $\frac{1}{3}$ para os estados 6K, 4K, e 2K respectivamente. Representando assim, o percentual da capacidade total que está disponível no *cluster*. A taxa de recompensa para todos os outros estados é 0 porque o *cluster* está em falha, assim não há núcleos de processador disponíveis.

A Tabela 5.6 mostra os resultados deste estudo, considerando a disponibilidade em estado estacionário para sistema redundante e não redundante, disponibilidade orientada à capacidade (COA) e *downtime* anual por hora para cada arquitetura. O valor da disponibilidade orientada à capacidade (COA) - não apresentados na Tabela 5.6 é de 0,995% para todas as arquiteturas, pois o fracasso do Subsistema de nuvem não afeta esta métrica, e os *clusters* adicionados de uma arquitetura de *hardware* para outro são idênticos um ao outro. O resultado da média da capacidade disponível (*average available capacity - AAC*) variará de acordo com o tamanho de cada arquitetura.

A existência de um único ponto de falha explica os maus resultados para as disponibi-

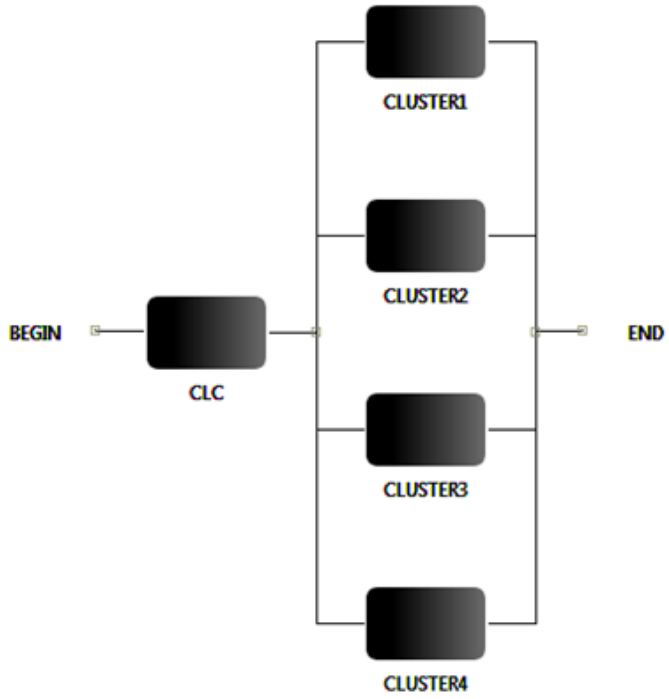


Figura 5.13: Cenário IV

Tabela 5.6: Medidas para Todas as Arquiteturas e suas Variantes

Arquitetura	Disponibilidade		Downtime (h)		AAC
	Não-Redund.	Redund.	Não-Redund.	Redund.	
A1	99,439671	99,716768	49,08	24,81 h	5.97
A2	99,718659	99,996533	24,64	0,30 h	11.94
A3	99,719441	99,997318	24,58	0,23 h	17.91
A4	99,719443	99,997320	24,58	0,23 h	23.88
A5	99,719443	99,997320	24,58	0,23 h	29.85

lidade de estado estacionário para o sistema não redundante, enquanto que os resultados para o sistema redundante mostram que uma simples replicação em um componente crítico pode aumentar a disponibilidade do sistema, reduzindo o tempo em uma significativa percentagem (superior a 99% de redução). Por outro lado, a adição dos *clusters* em ambas as arquiteturas não gera grandes contribuições. Note-se que os valores de disponibilidade entre cenários III, IV e V estão muito próximas entre si. Aumentar o número de clusters neste tipo de sistema de nuvem privada só se justifica pela capacidade de aceitar as cargas de trabalho maiores, ou seja, valores maiores de instâncias de VM. Os resultados indicam que AAC a eventual ocorrência de falhas tem um impacto significativo sobre a capacidade disponível, uma vez que em todos os cenários do AAC é perto da sua capacidade máxima.

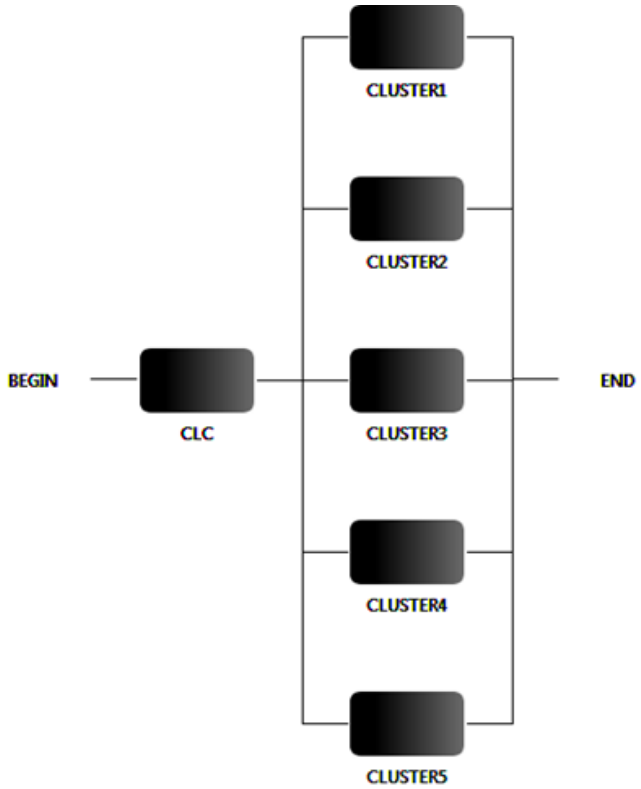


Figura 5.14: Cenário V

Pode ser visto que a arquitetura A5 tem a melhor AAC, mas, por outro lado, apresenta um tempo de inatividade anual de cerca de 24 horas, quando não há nenhuma redundância no Subsistema de nuvem para contornar eventuais falhas.

Portanto, a análise dos modelos propostos mostram que a adição de mais *clusters* para ambientes de nuvem privada, além de 3, é uma ação que só faz sentido se o objetivo é aumentar a capacidade. Se o objetivo é ter alta disponibilidade, é necessário e suficiente para investir em redundância para os Subsistemas de nuvem, ou até mesmo empregar outros mecanismos de tolerância a falhas, não estudados neste trabalho.

5.4 Considerações Finais

Este capítulo apresentou estudos de caso de dependabilidade de infraestruturas de nuvem privada com base no sistema *Eucalyptus*. Neste capítulo, cenários foram apresentados e três estudos de caso foram realizados. Inicialmente, um estudo de caso de disponibilidade de uma arquitetura considerada simples levou em conta a aplicabilidade de um mecanismo de redundância *warm-standby* no componente mais crítico do sistema. O objetivo desse

estudo é verificar o impacto que tal mecanismo traz para uma arquitetura em nuvem. Em seguida, um outro estudo de caso é apresentado na tentativa de desenvolver diversas outras arquiteturas de nuvem privada, utilizando componentes *Eucalyptus* separados, divididos em Subsistemas. O objetivo desse estudo é comparar a disponibilidade das arquiteturas propostas, aplicando redundância em diversos pontos da arquitetura, verificando disponibilidade e custo de aquisição de componentes. Finalmente, o terceiro estudo de caso consiste em verificar a disponibilidade no estado estacionário e a disponibilidade orientada à capacidade das arquiteturas propostas.

6

Conclusões e Trabalhos Futuros

Você nunca sabe que resultados virão da sua ação.
Mas se você não fizer nada, não existirão resultados.
—MAHATMA GANDHI

O provisionamento de serviços em um ambiente de computação em nuvem requer alta disponibilidade de hardware e componentes de software. A adição de componentes redundantes pode melhorar a disponibilidade do sistema, provendo melhor qualidade na entrega do serviço.

As principais contribuições deste trabalho são as proposições de modelos hierárquicos para infraestrutura de computação em nuvem, a avaliação do impacto da aplicabilidade de um mecanismo de replicação *warm-standby* em pontos críticos do sistema de computação em nuvem, utilizando recursos em uma infraestrutura de nuvem baseada no *software Eucalyptus*.

Com o intuito de avaliar os modelos propostos, foram realizados três estudos de caso. O primeiro estudo teve o objetivo de avaliar o impacto da disponibilidade, comparando os resultados de um sistema redundante e outro não redundante. Os resultados mostram um aumento na disponibilidade do sistema proposto redundante, evidenciada pelo acréscimo de dois noves para quatro noves, bem como uma diminuição do *downtime* anual de 46 horas para apenas 43 minutos. Os resultados também demonstram que a simples substituição de *hardware* por máquinas mais confiáveis não irá produzir uma melhoria tão grande na disponibilidade do sistema. No segundo estudo de caso, foi considerada a divisão dos componentes do *Eucalyptus* em subsistemas. Cada componente do *Eucalyptus* foi modelado de modo que os componentes de *software* fossem instalados e configurados em máquinas distintas, subdividindo a arquitetura em Subsistemas de

Nuvem, *cluster* e Nós. Aqui, observou-se o custo de construção de uma arquitetura para ambiente de computação em nuvem, bem como a disponibilidade do sistema. Foi aplicada redundância em duas partes do sistema (Subsistema da Nuvem e de *cluster*) e comparados os resultados. Um único ponto de entrada de clientes na nuvem é o motivo de maus resultados das arquiteturas não redundantes. O último estudo de caso analisou a disponibilidade orientada à capacidade (COA) das arquiteturas propostas no segundo estudo de caso. Os resultados mostram que, quando se tem poder computacional, os resultados são bons em se comparando com a disponibilidade no estado estacionário. Os resultados obtidos através dos modelos propostos demonstram a grande aplicabilidade em análise de dependabilidade de sistemas em nuvem, visto que a avaliação desses cenários sem a utilização dos modelos seria uma tarefa complexa, dispendiosa financeiramente e custosa.

6.1 Contribuições

Como resultado do trabalho apresentado nesta dissertação, as seguintes contribuições podem ser destacadas:

- Benefícios de um mecanismo de replicação *warm-standby* em um ambiente de computação em nuvem. Um sistema composto de um controlador geral GC e cinco nós é utilizado para análise dos resultados. Aplicando redundância no GC, é possível verificar o grande aumento de disponibilidade do sistema.
- A divisão da arquitetura em subsistemas permite criar várias arquiteturas e observar o impacto da disponibilidade do sistema e o custo de aquisição de máquinas, bem como avaliar a capacidade de fornecimento de recursos da arquitetura proposta.

Além da contribuição mencionada, alguns artigos que apresentam os resultados desta dissertação foram produzidos:

1. *Jamilson Dantas, Rubens Matos, Jean Araujo, and Paulo Maciel, "An availability model for eucalyptus platform: An analysis of warm-standy replication mechanism,"in The 2012 IEEE International Conference on Systems, Man, and Cybernetics (IEEE SMC 2012), Seoul, Korea, 2012.*
2. *Jamilson Dantas, Rubens Matos, Jean Araujo and Paulo Maciel, "Models for Dependability Analysis of Cloud Computing Architectures for Eucalyptus Platform,"International Transactions on Systems Science and Applications. December 2012.*

- Trabalhos a serem publicados.

1. *Jamilson Dantas, Rubens Matos, Jean Araujo and Paulo Maciel, "Availability Modeling and Analysis of Cloud Computing Architectures for Eucalyptus Platform".*

6.2 Trabalhos Futuros

Como trabalho futuro, pretende-se reproduzir alguns cenários propostos, com a aplicabilidade das abordagens mencionadas, a fim de realizar experimentos *Testbed*, verificando a consistência de dados entre os servidores replicados.

A utilização dos *softwares, Heartbeat* e *DRBD* auxiliando no processo de ativação e cópias de dados nos servidores replicados, garantindo a confiabilidade da abordagem da redundância *warm-standy*.

Referências Bibliográficas

[1] Amazon (2011). Amazon Elastic Block Store (EBS). Amazon.com, Inc. Available in: http://aws.amazon.com/ebs.

[2] Amazon (2012a). Amazon - amazon cloud drive. Amazon. Available in: http://www.amazon.com/.

[3] Amazon (2012b). Amazon elastic compute cloud - ec2. Amazon. Available in: http://aws.amazon.com/pt/ec2/.

[4] app engine, G. (2012). Google app engine. Google.com. Available in: https://developers.google.com/appengine/.

[5] Araújo, C. (2009). Avaliação e modelagem de desempenho para planejamento de capacidade do sistema de transferência eletrônica de fundos utilizando tráfego em rajada.

[6] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., *et al.* (2010a). Above the clouds: A view of cloud computing. Technical report, Technical report.

[7] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010b). A view of cloud computing. *Commun. ACM*, **53**(4), 50–58.

[8] Avizienis, A., Laprie, J., Randell, B., *et al.* (2001). Fundamental concepts of dependability. *Technical Report Series-University Of Newcastle Upon Tyne Computing Science*.

[9] Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. E. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Sec. Comput.*, **1**(1), 11–33.

[10] Azure, W. (2012). Windows azure - uma plataforma sólida na nuven para Ídeias criativas. Windows Azure. Available in: http://www.windowsazure.com/pt-br/.

[11] Bolch, G., Greiner, S., de Meer, H., and Trivedi, K. (2006). *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley-Interscience.

[12] Callou, G., Maciel, P., Tutsch, D., , and Araújo, J. (2012). Models for dependability and sustainability analysis of data center cooling architectures. In *The 2nd International Workshop on Dependability of Clouds, Data Centers and Virtual Machine Technology (DCDV 2012) in conjunction with The 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, Boston, MA, USA.

[13] Carolan, J. and Gaede, S. (2009). Introduction to cloud computing architecture. *SUN Microsystems Inc., pp.–1-40*.

[14] Cassandras, C. and Lafortune, S. (1999). *Introduction to discrete event systems*, volume 11. Kluwer academic publishers.

[15] Chaudhary, V., Cha, M., Walters, J., Guercio, S., and Gallo, S. (2008). A comparison of virtualization technologies for hpc. In *Advanced Information Networking and Applications (AINA 2008). 22nd Int. Conf. on*, pages 861–868.

[16] Chen, R. and Bastani, F. (1994). Warm standby in hierarchically structured process-control programs. *Software Engineering, IEEE Transactions on*, **20**(8), 658–663.

[17] Chuob, S., Pokharel, M., and Park, J. S. (2011). 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*, pages 289 –296.

[18] Company, H.-P. D. (2006). Redundant array of independent disks (raid) on hp compaq dc5750 business pcs.

[19] Corporation, M. (2012). Windows azure. Microsoft. Available in: https://www.windowsazure.com/pt-br/.

[20] D, J., Murari, K., Raju, M., RB, S., and Girikumar, Y. (2010). *Eucalyptus Beginner's Guide*, uec edition.

[21] de Araújo, J. C. T. (2012). *Software Aging Monitoring Strategies and Rejuvenation Policies for Eucalyptus Cloud Computing Platform*. Master's thesis, Universidade Federal de Pernambuco.

[22] de Sousa, E. (2009). Avaliação do impacto de uma política de manutenção na performabilidade de sistemas de transferência eletrônica de fundos.

[23] Dell (2012). Dell computers. Available in: http://www.dell.com/.

[24] Dillon, T., Wu, C., and Chang, E. (2010). Cloud computing: Issues and challenges. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 27–33. IEEE.

[25] DRBD (2012). Distributed replicated block device. http://www.drbd.org/.

[26] Ebeling, C. (2004). *An introduction to reliability and maintainability engineering*. Tata McGraw-Hill Education.

[27] Eucalyptus (2009). *Eucalyptus Open-Source Cloud Computing Infrastructure - An Overview*. Eucalyptus Systems, Inc., Goleta, CA.

[28] Eucalyptus (2010a). *Cloud Computing and Open Source: IT Climatology is Born*. Eucalyptus Systems, Inc., Goleta, CA.

[29] Eucalyptus (2010b). Eucalyptus cloud computing platform - administrator guide. Technical report, Eucalyptus Systems, Inc. Version 1.6.

[30] Eucalyptus (2012). Eucalyptus - the open source cloud platform. Eucalyptus Systems. Available in: http://open.eucalyptus.com/.

[31] Florin, G. and Natkin, S. (1989). Matrix product form solution for closed synchronized queuing networks. In *Petri Nets and Performance Models, 1989. PNPM89., Proceedings of the Third International Workshop on*, pages 29–37. IEEE.

[32] Furht, B. and Escalante, A. (2010). *Handbook of cloud computing*. Springer.

[33] GoGrid (2012). Gogrid cloud hosting. GoGrid.com. Available in: http://www.gogrid.com/.

[34] Gong, C., Liu, J., Zhang, Q., Chen, H., and Gong, Z. (2010). The characteristics of cloud computing. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 275–279. IEEE.

[35] Google (2012a). Google compute engine. Google. Available in: https://cloud.google.com/products/compute-engine.

[36] Google (2012b). Google drive. Google.com. Available in: https://drive.google.com/.

[37] Google (2012c). Serviços de e-mail do google - gmail.com. Gmail.com. Available in: http://www.gmail.com.

[38] Guimarães, A., Maciel, P., Matos Jr, R., and Camboim, K. (2011). Dependability analysis in redundant communication networks using reliability importance. In *Proc. of the 2011 Int. Conf. on Information and Network Technology (ICINT 2011)*, Chennai.

[39] Heartbeat (2012). Linux-ha. Available in: http://www.linux-ha.org.

[40] Heimann, D., Mittal, N., and Trivedi, K. (1990). Availability and reliability modeling for computer systems. *Advances in Computers*, **31**, 175–233.

[41] Herzog, U. (2001). Formal methods for performance evaluation. *Lectures on Formal Methods and PerformanceAnalysis*, pages 1–37.

[42] Hong, Z., Wang, Y., and Shi, M. (2012). Ctmc-based availability analysis of cluster system with multiple nodes. In *Advances in Future Computer and Control Systems*, volume 160, pages 121–125.

[43] Hu, T., Guo, M., Guo, S., Ozaki, H., Zheng, L., Ota, K., and Dong, M. (2010). Mttf of composite web services. In *Parallel and Distributed Processing with Applications (ISPA), 2010 Int. Symp. on*, pages 130 –137.

[44] IBM (2012). Ibm smartcloud - the cloud enterprises trust. IBM. Available in: http://www.ibm.com/cloud-computing/us/en/.

[45] Kim, D. S., Machida, F., and Trivedi, K. (2009). Availability modeling and analysis of a virtualized system. In *Dependable Computing, 2009. PRDC '09. 15th IEEE Pacific Rim Int. Symp. on*, pages 365–371.

[46] Kuo, W. and Zuo, M. (2002). *Optimal reliability modeling: principles and applications*. Wiley.

[47] Leangsuksun, C., Shen, L., Liu, T., Song, H., and Scott, S. (2003). Availability prediction and modeling of high availability oscar cluster. In *Proceedings of the IEEE International Conference on Cluster Computing (Cluster 2003)*, Hong Kong.

[48] Leangsuksun, C. B., Shen, L., Liu, T., and Scott, S. L. (2005). Achieving high availability and performance computing with an ha-oscar cluster. *Future Generation Computer Systems*, **21**(4), 597 – 606.

[49] Liu, T. and Song, H. (2003). Dependability prediction of high availability oscar cluster server. In *Proceedings of the 2003 Int. Conf. on Parallel and Distributed Processing Techniques and Applications*.

[50] Maciel, P., Trivedi, K. S., Matias, R., and Kim, D. S. (2011). Dependability modeling. In *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*. IGI Global, Hershey.

[51] Malhotra, M. (1994). Power-hierarchy of dependability model types. *IEEE Trans. on Reliability*, **43**(2), 493–502.

[52] Marwah, M., Maciel, P., Shah, A., Sharma, R., Christian, T., Almeida, V., Araújo, C., Souza, E., Callou, G., Silva, B., Galdino, S., and Pires, J. (2010). Quantifying the sustainability impact of data center availability. *SIGMETRICS Perform. Eval. Rev.*, **37**, 64–68.

[53] Matos Junior, R., Guimaraes, A., Camboim, K., Maciel, P., and Trivedi, K. (2011). Sensitivity analysis of availability of redundancy in computer networks. In *Proc. of The Fourth Int. Conf. on Communication Theory, Reliability, and Quality of Service (CTRQ 2011)*, Budapest.

[54] Mell, P. and Grance, T. (2011). The nist definition of cloud computing (draft). *NIST special publication*, **800**, 145.

[55] Musa, J. (2004). *Software reliability engineering: more reliable software, faster and cheaper*. Tata McGraw-Hill Education.

[56] Nimbus (2012). Nimbus - nimbus is cloud computing for science. Nimbus. Available in: http://www.nimbusproject.org/.

[57] OpenNebula (2012). Opennebula - the open source solution for data center virtualization. OpenNebula. Available in: http://opennebula.org/.

[58] Oracle (2012). Oracle cloud - sun cloud computing resource kit, software. hardware. complete. Oracle. Available in: http://www.oracle.com/.

[59] P. R. M. Maciel, K. Trivedi, R. M. J. and Kim, D. (2010). Dependability modeling in: Performance and dependability in service computing: Concepts, techniques and research directions. *Ed. Hershey: IGI Global*.

[60] Parhami, B. (1988). From defects to failures: a view of dependable computing. *ACM SIGARCH Computer Architecture News*, **16**(4), 157–168.

[61] Rausand, M. and Høyland, A. (2003). *System reliability theory: models, statistical methods, and applications*, volume 396. Wiley-Interscience.

[62] Reisner, P. (2002). Drbd - distributed replicated block device. In *Proc. of the 9th Int. Linux System Technology Conference*, Cologne.

[63] Resnick, R. (1996). A modern taxonomy of high availability.

[64] Sahner, R., Trivedi, K., and Puliafito, A. (1996). *Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package*. Kluwer Academic Publishers.

[65] SalesForce (2012). The social enterprise platform. SalesForce.com. Available in: http://www.salesforce.com/platform.

[66] Sempolinski, P. and Thain, D. (2010). A comparison and critique of eucalyptus, opennebula and nimbus. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 417–426. Ieee.

[67] Sun, D., Chang, G., Guo, Q., Wang, C., and Wang, X. (2010). A dependability model to enhance security of cloud environment using systemlevel virtualization techniques. In *Proc. First Int. Conf. on Pervasive Computing, Signal Processing and Applications (PCSPA 2010)*, Harbin.

[68] SunMicrosystems (2009). *Introduction to Cloud Computing Architecture*. Sun Microsystems, Inc., 1 edition.

[69] Trivedi, K., Hunter, S., Garg, S., and Fricks, R. (1996). Reliability analysis techniques explored through a communication network example. In *International Workshop on Computer-Aided Design, Test, and Evaluation for Dependability*, pages 2–3.

[70] Trivedi, K., Vasireddy, R., Trindade, D., Nathan, S., and Castro, R. (2006). Modeling high availability systems. In *Proc. Pacific Rim Dependability Conference*, pages 11–20.

[71] Trivedi, K., Kim, D., Roy, A., and Medhi, D. (2009). Dependability and security models. In *Design of Reliable Communication Networks, 2009. DRCN 2009. 7th International Workshop on*, pages 11–20. IEEE.

[72] Veríssimo, P. and Rodrigues, L. (2001). Fundamental security concepts. *Distributed Systems for System Architects*, pages 377–393.

[73] Wan, Y., Feng, D., Yang, T., Deng, Z., and Liu, L. (2008). The adaptive heartbeat design of high availability raid dual-controller. In *Multimedia and Ubiquitous Engineering, 2008. MUE 2008. International Conference on*, pages 45–50. IEEE.

[74] Weber, T. (2003). Tolerância a falhas: conceitos e exemplos. *Programa de Pós-Graduação em Computação–Instituto de Informática–UFRGS*.

[75] Wei, B., Lin, C., and Kong, X. (2011). Dependability modeling and analysis for the virtual data center of cloud computing. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pages 784–789. IEEE.

[76] Yeow, W.-L., Westphal, C., and Kozat, U. (2010). A resilient architecture for automated fault tolerance in virtualized data centers. In *IEEE Network Operations and Management Symposium (NOMS)*, pages 866 –869.

[77] Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, **1**(1), 7–18.