Pós-Graduação em Ciência da Computação

Carlos Alexandre Silva de Melo

AVALIAÇÃO DA DISPONIBILIDADE DE INFRAESTRUTURA DE SINCRONIZAÇÃO DE DADOS

Dissertação de Mestrado

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

RECIFE
2016

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

Carlos Alexandre Silva de Melo

AVALIAÇÃO DA DISPONIBILIDADE DE INFRAESTRUTURA DE SINCRONIZAÇÃO DE DADOS

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

Orientador: *Paulo Romero Martins Maciel*

RECIFE
2016

Carlos Alexandre Silva de Melo
Avaliação da Disponibilidade de Infraestrutura de Sincronização de Dados/ Carlos Alexandre Silva de Melo. – RECIFE, 2016-
101 p. : il. (algumas color.) ; 30 cm.

Orientador Paulo Romero Martins Maciel

Dissertação de Mestrado – Universidade Federal de Pernambuco, 2016.

1. Palavra-chave 1: Disponibilidade. 2. Palavra-chave 2: Sincronização de Dados. 3. Palavra-chave 3: Modelos estocásticos. 4. Palavra-chave 4: Disponibilidade Orientada a Capacidade. 5. Palavra-chave 5: Análise de Custos de Implantação. 6. Palavra-chave 6: Computação em Nuvem. I. Orientador: Paulo Romero Martins Maciel. II. Universidade Federal de Pernambuco. III. Avaliação da Disponibilidade de Infraestrutura de Sincronização de Dados

CDU 02:141:005.7

Dissertação de mestrado apresentada por **Carlos Alexandre Silva de Melo** ao programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **Avaliação da Disponibilidade de Infraestrutura de Sincronização de Dados**, orientada pelo **Prof. Paulo Romero Martins Maciel**:

Prof. Paulo Romero Martins Maciel
Centro de Informática/UFPE

Prof. Eduardo Antônio Guimarães Tavares
Centro de Informática/UFPE

Bruno Silva
IBM Research Brazil.

RECIFE
2016

Dedico esta dissertação a Deus e a minha família por estarem sempre ao meu lado.

Agradecimentos

Gostaria de agradecer a minha família, em especial aos meus pais e a Raquel, mulher amável que esteve ao meu lado em todos os momentos e me auxiliou em todos os passos dessa jornada.

Ao Professor Paulo Maciel por orientar-me no decorrer desta jornada e aos amigos do grupo MoDCS, em especial a Jamilson, Jean e Danilo pelo suporte a esta pesquisa. Aleciano e Charles que batalharam ao meu lado todos os dias, ao longo destes dois anos. Micael, Emanoel, José Antônio e Valter por, assim como eu, terem encarado a vida no apartamento 04. Ygor, César, Gleydson, David, Brenda e Nayá pelos momentos de descontração e pelo apoio incondicional.

Greed can be a very powerful ally.

—QUI-GON JINN

Resumo

Dispositivos eletrônicos como celulares, *smartphones*, televisores, *smartwatches*, computadores pessoais, *laptops*, consoles de mesa e portáteis estão, de algum modo, em praticamente todos os lares que se visite nos últimos anos, e é bem provável que pelo menos um deles possua conexão com a Internet e/ou algum tipo de mecanismo de sincronização instalado. Já a adoção de mecanismos de sincronização, como Dropbox, Google Drive e One Drive, tornou-se de grande importância para usuários de sistemas computacionais, que almejam ter acesso aos seus dados pessoais, sempre que possível, de forma justa e confiável, 24 horas por dia, os 07 dias da semana. E é este tipo de exigência que proporciona um aumento da demanda e da qualidade do serviço oferecido pelas prestadoras uma vez que, com a propagação do paradigma de computação em nuvem, vem sendo migrado em larga escala para este tipo de plataforma. A adoção da computação em nuvem é ligada diretamente à firmação de contrato entre prestadora de serviço e fornecedora de infraestrutura. Este contrato, também conhecido como Acordo de Nível de Serviço ou Service Level Agreements (SLA), prevê o pagamento de multas, caso os valores de disponibilidade estabelecidos não sejam alcançados. Quanto maior os valores de disponibilidade requeridos, mais alto será o custo da arquitetura que proverá o serviço, sendo imprescindível uma análise, meticulosa, antes de qualquer investimento realizado. Sendo uma possibilidade para os que desejam fornecer serviços, através da Internet, arcar com os custos de implantação de sua própria infraestrutura computacional. A compra das máquinas e equipamentos necessários para a implantação de uma infraestrutura própria pode ter um custo menor que a realização de um SLA com alguma grande empresa especializada no provimento de infraestruturas de computação em nuvem como Amazon e Google. Este trabalho propõe modelos hierárquicos baseados em RBDs, SPNs e CTMCs para avaliar disponibilidade e custos de implantação de serviços de sincronização de dados hospedados em plataformas privadas de computação em nuvem. Os modelos propostos poderão ser utilizados como subsídio para o planejamento de infraestruturas que possam prover serviços de sincronização de dados, e auxiliar nos processos internos de tomada de decisão e análise entre custo $\times$ benefício para determinação do melhor ambiente.

Palavras-chave: Sincronização de dados, Computação em Nuvem, Modelos Hierárquicos, RBDs, SPNs, CTMCs

Abstract

Electronic devices such as mobile phones, smartphones, televisions, smartwatches, PCs, laptops, table consoles and handhelds are present in some amount in virtually every home you've visited in the past few years, and it's likely that at least one of them has an Internet connection and some synchronization mechanism installed. Since the adoption of synchronization mechanisms, such as Dropbox, Google Drive, and One Drive, became of great importance to computer systems users, which aims to gain access to your personal data wherever possible, fairly and reliably, 24 hours a day, seven days a week. This kind of requirement provides an increase in demand and the quality of service offered by the providers. This service, with the spread of cloud computing paradigm, has been migrated in large scale for this type of platform. The adoption of cloud computing is linked directly to the contract affirmation between the service provider and supplier of infrastructure, this contract, also known as Service Level Agreement (SLA) provides for the payment of penalties if the availability values set are not achieved. The higher the values of desired availability, the higher the cost of the architecture that will provide the service, being indispensable a thorough analysis before any investment. It is a possibility for those who wish to serve over the Internet, bear the implementation costs of their computing infrastructure, responsible for providing the service. The purchase of machinery and equipment required for the deployment of its infrastructure may have a lower cost than the fulfillment of a contract with some big specialized company. This research proposes hierarchical models, based in RBDs, SPNs, and CTMCs for availability evaluation and deployment costs for data synchronization services hosted in a private cloud computing platform. The proposed model can be used as input for the planning of infrastructure that can provide data synchronization services, and assist in the internal decision-making and cost $\times$ benefit analysis determine which one is the best environment.

Keywords: Synchronization, Cloud Computing, Hierarchical Models, RBD, SPN, CTMC

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

SyncML Synchronization Markup Language 16

CTMC Continuous Time Markov Chain 16

RBD Reliability Block Diagram 16

XML eXtensible Markup Language 16

MTTF Mean Time to Failure 26

MTTR Mean Time to Repair 26

OMA Open Mobile Alliance 22

SLA *Service Level Agreements* 16

SPN Stochastic Petri Nets 16

VoD Video on Demand 19

MoDCS Modeling of Distributed and Concurrent Systems 86

EFM Energy Flow Models 86

OMA Open Mobile Alliance 22

PIM Personal Information Management 22

VPN Virtual Private Networks 38

MTTA Mean Time To Activate 55

IaaS Infraestrutura como Serviço 36

PaaS Plataforma como Serviço 36

SaaS Software como Serviço 36

CLC Controlador de Nuvem 38

CC Controlador de Cluster 38

SC Controlador de Armazenamento 38

NC Controlador de Nó 39

CRC Cyclic Redundancy Check 15

MD5 Message-Digest algorithm 5 15

Sumário

1

Introdução

Sincronizar, ato de tornar simultâneo FERREIRA (2014). Em sistemas de computadores, trata-se da manutenção da consistência de um conjunto de dados distribuídos entre vários dispositivos, como celulares, computadores pessoais e consoles portáteis LEE; LEE; KIM (2007).

Sob a perspectiva da sincronização de dados, cada dispositivo é responsável pela manutenção de uma cópia, ou réplica dos dados pertencentes a um usuário, grupo de usuários, empresa ou instituição TRAUD et al. (2008). Técnicas destinadas à sincronização de dados foram desenvolvidas com o passar dos anos, objetivando uma maneira mais automatizada, segura e eficaz de oferecer serviços de sincronização. Estas técnicas se subdividiram em abordagens otimistas e pessimistas.

Em 1980, início da corrida pelo desenvolvimento dos computadores pessoais e últimos momentos da já duradoura Guerra Fria, abordagens conhecidas como pessimistas foram frequentemente utilizadas, neste tipo de abordagem os administradores de sistema eram responsáveis por definir uma réplica-chave para manter as demais cópias de um dado sempre atualizadas, esta réplica ficava hospedada em um dispositivo de armazenamento e se comunicava com as outras réplicas para propagar as atualizações mais recentes SAITO; SHAPIRO (2005). Se, por algum motivo, a réplica-chave fosse perdida mediante corrupção ou por problemas na unidade de armazenamento, um algoritmo realizava a comparação entre as réplicas remanescentes para determinar uma nova réplica para atuar como chave SAITO; SHAPIRO (2005).

Abordagens pessimistas são bem vistas para uma rede local de computadores, onde manter e analisar dados são tarefas de baixo custo computacional, porém tal proposta torna-se impraticável quando temos dispositivos espalhados por todo o globo terrestre e que requerem a realização da sincronização em um curto espaço de tempo, o que não é possível em uma abordagem pessimista, já que nesta abordagem intervalos de tempo mais longos seriam necessários, tornando os dados indisponíveis e inacessíveis por um longo período de tempo.

Para tentar mitigar as limitações enfrentadas por abordagens pessimistas, surgiram as então chamadas de otimistas, com o objetivo de transpor as barreiras relacionadas à distância entre dispositivos dentro de uma rede de computadores, melhorando a disponibilidade dos dados

e não dependendo de uma única réplica central como na abordagem pessimista, aplicando as atualizações quase que simultaneamente a todas as cópias distribuídas de um mesmo dado. Porém, sua maior vantagem também pode ser interpretada como desvantagem, já que para manter os dados atualizados através de um grande número de fontes à ocorrência de conflitos, concorrência e, por vezes, corrupção dos dados, torna-se mais frequente SAITO; SHAPIRO (2005).

Esta tríade (Concorrência, Conflitos e Corrupção) tende a ser ainda mais frequente em redes de maior complexidade, onde há um grande número de dispositivos interconectados, este é o caso da rede mundial de computadores, que traz consigo além de um gigantesco número de computadores pessoais, um ainda maior de dispositivos móveis; Estes que, atualmente, encontram-se em uma quantidade maior que a de seres humanos no mundo BOREN (2014). Algumas técnicas podem ser utilizadas para mitigar o impacto causado pela tríade nos arquivos e na comunicação entre os dispositivos distribuídos, como é o caso da *Cyclic Redundancy Check (CRC)* que consiste em um algoritmo de validação dos blocos de dados enviados através da aplicação de uma divisão polinomial PETERSON; BROWN (1961) e o *Message-Digest algorithm 5 (MD5)*, utilizado em grande escala na verificação de integridade de dados que podem ter sido corrompidos de modo não intencional MAO; CHEN; XU (2009). Mas, mesmo com a utilização destas técnicas, o crescimento no desenvolvimento de dispositivos móveis, que hoje é cinco vezes maior que a da nossa espécie BOREN (2014), fez emergir a necessidade de uma padronização para melhor interação entre os diversos tipos de dispositivos, incluso a sincronização de dados. Para tanto, *frameworks* foram desenvolvidos, visando uma redução na ocorrência dos conflitos e aplicáveis a dispositivos com baixo poder de processamento e grandes limitações na largura de banda, tanto para o envio, quanto para o recebimento de dados.

No geral, estes *frameworks* se encaixam nas arquiteturas do tipo cliente-servidor, onde uma aplicação em execução do lado do usuário interage com outra aplicação sendo executada do lado do servidor, realizando a comunicação e troca dos dados que serão sincronizados. No passado, estes servidores eram interpretados como máquinas de grande poder computacional e de alto custo, hoje, graças ao paradigma de computação em nuvem, houve uma grande mudança de perspectiva. Este poderoso paradigma proveniente da computação distribuída, caracteriza-se pela abstração dos recursos de *hardware* e *software* por parte de quem os consome, sendo então virtualizados, dinamicamente alocados, de grande capacidade de gerenciamento e onde armazenamento, plataformas e serviços são oferecidos sob demanda através da Internet FOSTER et al. (2008). Deste modo, aplicações que tenham caráter de servidoras, podem ser hospedadas como serviços em plataformas de computação em nuvem de menor custo, porém, deles, assim como de outros serviços, espera-se o maior valor de disponibilidade possível e que sejam providos durante 24 horas por dia, 7 dias por semana, ou os usuários ficarão impossibilitados de realizar a sincronia em um período de indisponibilidade.

Infraestruturas de computação em nuvem podem ser alugadas a provedoras, como Amazon, Google, HP e Microsoft, capazes de fornecer serviços através da Internet. Porém, estas

empresas estão sujeitas ao pagamento de multas caso desrespeitem um contrato previamente estabelecido: O *Service Level Agreements (SLA)*. Este contrato determina a qualidade do serviço que está sendo disponibilizado e o percentual de disponibilidade que o mesmo terá em um período de tempo pré-estabelecido HAUCK et al. (2009). Por isso, torna-se de suma importância uma estimativa prévia dos valores de disponibilidade deste tipo de ambiente computacional e o quanto iremos gastar para alcançá-los, levando-se em conta possíveis perdas financeiras e o impacto negativo à imagem da empresa, caso aquilo que fora previamente acordado seja desrespeitado.

Além do aluguel de infraestruturas às empresas, também é possível arcar com os custos necessários para à implementação de um ambiente próprio de computação em nuvem, oferecendo nela seus próprios serviços sem a obrigatoriedade do estabelecimento de um SLA, e a possibilidade da redução de custos durante sua implantação através da utilização de *softwares* e ferramentas *open source*.

Uma já conhecida especificação aberta que se enquadra na arquitetura cliente-servidor, adotada pela maioria dos sincronizadores, e que pode ser facilmente adaptada a um ambiente de computação em nuvem é o Synchronization Markup Language (SyncML). O SyncML rege uma especificação para *frameworks* de sincronização de dados entre dispositivos conectados à Internet LEE; LEE; KIM (2007), tem sua implementação e descrição inteiramente baseados em eXtensible Markup Language (XML), sendo esta uma de suas principais características: A troca de mensagens em XML entre os dispositivos pertencentes a um usuário; tornando-o um protocolo multiplataforma, e de grande amplitude, já que o XML é um padrão estável, reconhecido e interpretado por diversos sistemas computacionais.

Em resumo, o trabalho realizado nesta dissertação consistiu na proposição de modelos hierárquicos destinados à avaliação de disponibilidade de um serviço de sincronização de dados, hospedado e gerenciado por uma plataforma privada de computação em nuvem. O custo de implantação para as arquiteturas propostas também foi avaliado, além da disponibilidade orientada à capacidade.

Reliability Block Diagram (RBD), Stochastic Petri Nets (SPN) e Continuous Time Markov Chain (CTMC) MACIEL et al. (2011) foram utilizados na criação dos modelos propostos e nos processos de análise e avaliação dos ambientes, enquanto que técnicas como a de Análise de Sensibilidade foram utilizadas na determinação dos componentes mais importantes do sistema, a fim de auxiliar na proposição de novas arquiteturas com um maior valor de disponibilidade e um melhor custo $\times$ benefício.

Objetivos

O objetivo computacional desta pesquisa é a avaliação da disponibilidade de um serviço de sincronização de dados hospedado em plataforma privada de computação em nuvem. Para tanto, modelos estocásticos dispostos de maneira hierárquica são utilizados e através destes mo-

delos, somos capazes de fornecer os subsídios necessários para o planejamento de infraestruturas, bem como suas capacidades e custos de implantação. Já os objetivos específicos são listados abaixo:

- Elaboração de modelos que representem os modos operacionais da infraestrutura e do serviço de sincronização de dados;
- Validação dos modelos propostos através de experimentos;
- Aplicação de técnicas para a identificação dos componentes de maior impacto na disponibilidade da arquitetura básica;
- Proposição de novas arquiteturas, objetivando a identificação de melhorias na disponibilidade através da aplicação de redundância nos componentes que mais impactam a disponibilidade;

Para elaboração e definição dos objetivos apresentados, tornou-se necessária a etapa de levantamento bibliográfico de estudos já existentes e trabalhos correlatos àquilo que propomos. A Subseção 1.2 apresenta os principais trabalhos relacionados a esta pesquisa de mestrado e o quanto a nossa proposta se diferencia das demais.

Trabalhos relacionados

Uma série de trabalhos foi realizada nas áreas relacionadas a esta pesquisa, porém, nenhum deles engloba todas as características aqui propostas. Alguns avaliam métricas de desempenho para implementações de um sincronizador de dados *open source*, alguns avaliam a disponibilidade de outros serviços em plataformas de computação em nuvem, e ainda há aqueles que estudam o consumo energético e sua relação com o impacto ambiental causado por grandes infraestruturas computacionais, como *data centers*.

A busca por estes trabalhos se deu através do estabelecimento de algumas palavras-chave relacionadas ao SyncML como *sincronização de dados*, e outras à modelagem e avaliação de disponibilidade e métricas de dependabilidade em ambientes de computação em nuvem buscados em pares com a primeira *string* contendo *data synchronization* e a segunda podendo conter *stochastic model, deployment cost, power consumption, sensitivity analysis, availability* ou *performance*. O período de buscas ocorreu entre os meses de novembro de 2015 e janeiro de 2016, e limitou-se às principais bibliotecas digitais na área de ciência da computação: *ACM, ScienceDirect, Springer e IEEEXplore*. Como limitação de prazo, o intervalo definido para a pesquisa se estende entre os anos de 2010 e 2016.

Os trabalhos considerados de grande relevância para o trabalho aqui apresentado foram selecionados, considerando avaliações individuais e a sua relação com o tema proposto nesta dissertação. Em TIAN; LI (2012) uma implementação de um sincronizador baseado em SyncML

foi realizada, tendo este trabalho como principal propósito a avaliação de desempenho desta especificação. Os experimentos realizados consistiram na sincronia de dados de forma *Two-Way*, onde alterações no cliente são aplicadas ao servidor e vice-versa, este tipo de sincronização é um dos mais utilizados por provedoras deste tipo de serviço, os resultados obtidos comprovaram a eficiência e facticidade para esta tarefa em redes com e sem fio e apontaram uma maior eficiência nas operações de sincronização realizadas em redes cabeadas, onde 2000 arquivos foram sincronizados em trinta segundos, enquanto que na sem fio os mesmos 2000 arquivos precisaram de 75 segundos para serem enviados. A métrica de desempenho que comprovamos através deste trabalho é a disponibilidade orientada à capacidade, não estando interessados no tempo necessário para realização da sincronização de dados, apenas no estado de disponibilidade quanto aos componentes necessários para o provimento do serviço quando forem consultados.

Já em CHUN-LEI; JIAN-QIANG (2010), foi utilizada uma implementação de código aberto do SyncML, intitulada Funambol, desenvolvido inicialmente como Sync4j pela empresa norte-americana de mesmo nome e que, atualmente, trabalha com o provimento de sincronização de dados envolvendo plataformas de computação em nuvem Funambol. O principal objetivo do trabalho realizado por CHUN-LEI; JIAN-QIANG (2010) foi a promoção e o compartilhamento de bases de dados heterogêneas, contendo uma série de informações e conhecimentos pertinentes à medicina tradicional chinesa, produzindo uma maior integração entre médicos e pacientes do país com a maior população do mundo, através da sincronização de dados. Para a validação dos modelos de disponibilidade propostos por este trabalho, uma infraestrutura precisou ser configurada na plataforma de computação em nuvem, e a mesma implementação de código aberto (Funambol) foi utilizada.

LI; LI (2012) estuda a aplicação de um algoritmo para compressão de dados no processo de sincronização entre servidor e dispositivos móveis, visando uma redução no tempo total para realização desta operação. Este trabalho também utiliza da especificação SyncML e de uma implementação que segue suas diretrizes, tendo como principal diferencial dos trabalhos anteriores a codificação de Huffman, estas que substituem os trechos ou caracteres que se repetem dentro de um arquivo por um único símbolo. Experimentos foram realizados para comprovar a viabilidade da utilização deste tipo de técnica e que demonstram redução significativa nos tempos de sincronização, e no consumo de banda por parte do dispositivo móvel. Já nesta dissertação, os experimentos são utilizados para validação dos modelos propostos e consistem na injeção de falhas e execução de reparo dos principais componentes do serviço de sincronização de dados provido.

Já DANTAS et al. (2012b,a) propôs uma série de modelos destinados à avaliação de disponibilidade da plataforma de computação em nuvem Eucalyptus. Nestes trabalhos, SPNs, CTMCs e RBDs foram propostos no estabelecimento de uma arquitetura básica, além da avaliação dos efeitos e impactos da utilização de mecanismos de redundância do tipo *warm standby* para melhoria da disponibilidade da plataforma e como variações da arquitetura básica, considerando para a modelagem hierárquica e heterogênea os tempos de falha e reparo dos componentes

responsáveis pelo funcionamento do Eucalyptus, seguindo uma distribuição exponencial para estados e blocos. Mais tarde, em DANTAS et al. (2015) um estudo foi realizado visando uma análise de custos, consumo energético, proposição de modelos hierárquicos e determinação de equações para avaliação da disponibilidade orientada à capacidade para a implantação de uma nuvem pública com a plataforma Eucalyptus. Diferentemente da sequência de trabalhos apresentada por Dantas *et al* aqui avaliamos o impacto; não somente de redundâncias do tipo *warm standby*, mas também das do tipo *cold standby* e *hot standby* na disponibilidade da arquitetura, promovendo um conjunto de 13 arquiteturas variantes e realizando uma relação entre custo $\times$ *downtime* para implantação de cada uma delas.

BEZERRA et al. (2014); MELO et al. (2014) realizam a análise da disponibilidade de um sistema de Video on Demand (VoD), implantado em uma infraestrutura privada de computação em nuvem, gerida e controlada pela plataforma Eucalyptus. Modelos hierárquicos também foram utilizados, bem como a aplicação de técnicas para a análise de sensibilidade com o objetivo de determinar os componentes mais críticos do sistema e de técnicas de redundância para comprovar a melhoria na disponibilidade geral do serviço. A principal diferença entre esta dissertação e os trabalhos mostrados é o tipo de serviço oferecido, aqui avaliamos e determinamos o comportamento de um serviço de sincronização de dados, enquanto que nos anteriores um serviço de *streaming* de vídeo é avaliado; outra diferença em potencial são as arquiteturas propostas como variantes à arquitetura básica, além da técnica de análise de sensibilidade aplicada à arquitetura básica, outras propostas são exclusivas à esta dissertação como a avaliação de disponibilidade orientada à capacidade e a análise de custos para implantação e manutenção do serviço proposto.

Enquanto que CALLOU et al. (2012) propôs uma série de modelos destinados à quantificação do impacto ambiental que é proporcionado pelo consumo energético, bem como uma análise de custo e avaliações de métricas de dependabilidade para cinco *data centers* do mundo real. Já FERREIRA et al. (2015) demonstra um algoritmo balanceador de carga destinado à infraestruturas privadas de computação em nuvem, objetivando uma melhor distribuição energética entre os integrantes de uma infraestrutura, tomando como base a carga de trabalho recebida por cada um deles, determinando, assim, meios para a redução no impacto ambiental. Diferentemente destes trabalhos esta dissertação não provê modelos para a análise de consumo energético, porém realiza um levantamento dos equipamentos necessários para implantação de cada arquitetura proposta, além dos custos para cada equipamento e o consumo energético deles com base em seus respectivos manuais e especificações técnicas fornecidas pelos sites onde são vendidos.

No estudo realizado por YANG et al. (2016) uma abordagem para gerenciamento de metadados é descrita visando uma melhora no tempo e eficiência para acesso e compartilhamento de dados distribuídos, reduzindo a frequência de transmissão e a latência no acesso aos dados. Técnicas de redundância do tipo *warm standby* são utilizadas para reduzir o impacto das falhas na dependabilidade do ambiente de computação em nuvem, de forma semelhante à realizada neste trabalho, onde os subsistemas do nó ou do frontend são dispostos em redundância; a principal

diferença entre este trabalho e o proposto por esta dissertação é o fato de não serem utilizados modelos gráficos que representam o comportamento da plataforma de computação em nuvem.

A Tabela 1.1 resume as principais contribuições desta pesquisa e em que ela se diferencia dos demais trabalhos apresentados até aqui.

Tabela 1.1: Principais contribuições da dissertação

	Contexto	Infraestrutura de Sincronização de Dados	Uso de Modelos	Custos de Implantação	Consumo Energético	Análise de Sensibilidade	Métrica de Dependabilidade	Métrica de Desempenho
Esta dissertação	Computação em Nuvem	✓	✓	✓	✓	✓	✓	✓
TIAN; LI (2012)	Desempenho	✓						✓
CHUN-LEI; JIANG-QING (2010)	Medicina	✓						
LI; LI (2012)	Desempenho	✓						✓
DANTAS et al. (2012b)	Computação em Nuvem		✓				✓	
DANTAS et al. (2012a)	Computação em Nuvem		✓	✓			✓	
DANTAS et al. (2015)	Computação em Nuvem		✓	✓	✓		✓	✓
BEZERRA et al. (2014)	Computação em Nuvem		✓			✓	✓	
MELO et al. (2014)	Computação em Nuvem		✓			✓	✓	
CALLOU et al. (2012)	Data Center		✓	✓	✓			✓
FERREIRA et al. (2015)	Computação em Nuvem		✓		✓			
YANG et al. (2016)	Computação em Nuvem	✓					✓	✓

Esta Subseção teve por objetivo: esclarecer as principais diferenças entre esta dissertação e os trabalhos já existentes relacionados à sincronização de dados e a avaliação de disponibilidade de serviços hospedados em plataformas de computação em nuvem. Esta dissertação engloba um maior número de abordagens para determinar o comportamento e de métricas como avaliação de disponibilidade, consumo energético, análise de sensibilidade, disponibilidade orientada à capacidade e determinação de custos para implantação de serviços de sincronização de dados em infraestruturas privadas de computação em nuvem. Abordando novos cenários e sugerindo arquiteturas que proporcionem uma relação custo $\times$ benefício aos que planejam ofertar este tipo de serviço.

Dentre as principais contribuições deste trabalho estão os artefatos, estes que são compostos por modelos dispostos de maneira hierárquica gráfica (RBD, RBD e CTMC) e textuais (Scripts), podendo sofrer reajustes virtualmente infinitos, visando uma adequação à realidade econômica e as necessidades contratuais dos que almejam oferecer serviços de sincronização em uma plataforma privada de computação em nuvem. A avaliação de disponibilidade, disponibilidade orientada à capacidade e custos de implantação para uma infraestrutura computacional são

os principais resultados alcançados.

Estrutura da Dissertação

Esta dissertação está organizada na seguinte ordem: No Capítulo 2 são apresentados os principais conceitos utilizados como embasamento da pesquisa realizada, incluso computação em nuvem, sincronização de dados, análise de sensibilidade, bem como dependabilidade e as métricas e/ou técnicas associadas para a modelagem e avaliação deste termo, como RBD, SPN e CTMC. O Capítulo 3 apresenta a metodologia utilizada, introduzindo os modelos propostos para avaliação de disponibilidade das arquiteturas apresentadas destinadas ao fornecimento do serviço de sincronização de dados. Já o Capítulo 4 apresenta os estudos de caso e as variações de arquiteturas, com o objetivo único de avaliar a factividade dos artefatos apresentados no Capítulo anterior, através dos resultados obtidos por meio de análises dos modelos previamente propostos. O Capítulo 5 apresenta as conclusões obtidas por este trabalho, sumarizando as suas contribuições e apresentando uma série de possibilidades caracterizadas como trabalhos futuros.

2

Referencial Teórico

Este Capítulo tem por objetivo apresentar as principais tecnologias utilizadas no processo de desenvolvimento desta dissertação e em que foram aplicadas. A começar pelo tipo de serviço escolhido: sincronização de dados e, logo na sequência o termo dependabilidade e suas conexões são explanadas, bem como a definição da nossa principal métrica de interesse: disponibilidade. Na terceira Seção conceitos de modelagem são demonstrados, além da sua relação com a avaliação de métricas de dependabilidade. Em seguida, a definição de computação em nuvem é apresentada, este foi o tipo de ambiente escolhido por esta pesquisa. Por fim, técnicas de análise de sensibilidade são apresentadas, bem como a justificativa para sua utilização.

Sincronização de Dados

É o processo que permite a consistência de dados entre um dispositivo fonte e um mecanismo de armazenamento destino ou vice-versa AGARWAL; STAROBINSKI; TRACHTENBERG (2002). Muitos grupos surgiram com o objetivo de criar aplicações e ferramentas que proporcionassem serviços de sincronização de dados aos mais diversos nichos de usuários, é o caso do Dropbox, Google Drive e OneDrive, além destes, um antecessor intitulado de SyncML surgiu no início dos anos 2000.

SyncML

É um protocolo padrão atualmente definido pela Open Mobile Alliance (OMA) e que destina-se à sincronização de dados em dispositivos multiplataformas HORDE (2013) para trabalhar em arquiteturas do tipo cliente-servidor, onde todos os dispositivos conectados à rede poderão enviar e receber mensagens de sincronização SMOLAREK (2011); OMA (2012).

Em suas primeiras versões a especificação do SyncML determinava a elaboração de um *framework* que permitia a sincronia de informações e dados pessoais, ou simplesmente Personal Information Management (PIM) data. Exemplos rotineiros deste tipo de informação são as provenientes de eventos da agenda e dos contatos de um usuário, hoje, esta já reformulada

especificação foi acoplada ao gerenciador de dispositivo móveis desenvolvido pela OMA, e é capaz de entregar e colher dados provenientes dos mais diversos dispositivos, como mostra a Figura 2.1.

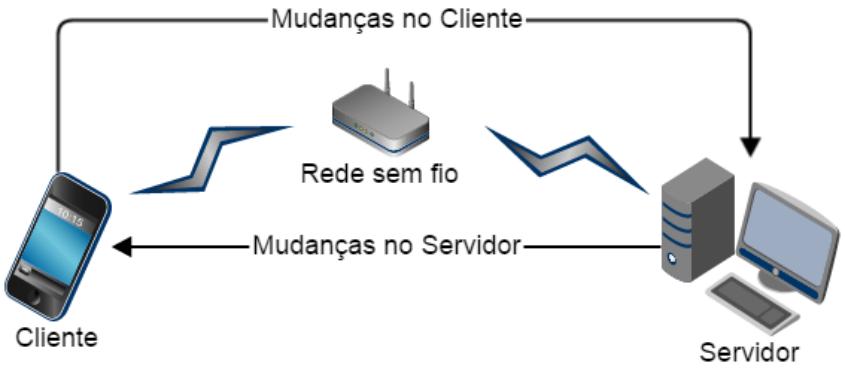


Figura 2.1: Comunicação entre aplicação cliente e servidor (baseado no modelo apresentado em FUNAMBOL (2010))

Com base em uma análise mais aprofundada, é possível determinar quais os principais componentes que garantem o funcionamento de um *framework* desenvolvido através desta especificação. Arquiteturas do tipo cliente-servidor necessitam de algo que permita a interação entre dois dispositivos, por vezes distintos, agindo como um *middleware*, já que permitirá a conexão da aplicação cliente com uma aplicação do tipo servidor, este é o *Web Server*, o SyncML adota como servidor web padrão o Apache.

Já para o armazenamento do caminho que leva aos arquivos de um determinado usuário, é necessária uma aplicação para gerenciamento de banco de dados (SGBD), esta que pode ser o MySQL, já renomado, adotado e reconhecido mundialmente.

E por fim, o não menos importante, servidor de sincronização, ele que realizará o envio e gestão dos dados entre a fonte e o destino, este que também será o responsável pela segurança no acesso à informação e contas de usuário.

Mas o que queremos avaliar através destas informações? Dados comportamentais do serviço que está sendo ofertado, e, para se determinar o comportamento deste serviço, utilizaremos de métricas e números relacionados ao termo **dependabilidade**.

Dependabilidade

Em AVIZIENIS et al. (2004) o termo dependabilidade é definido como a habilidade de um sistema computacional de entregar ou fornecer um serviço de maneira justa e confiável. Esta definição de alto nível considera a dependabilidade como uma grande caixa preta que só pode ser analisada do ponto de vista do usuário do sistema, e sua avaliação será dada com base no comportamento esperado desta caixa, e das respostas por ela fornecidas a cada entrada realizada.

Por tratar-se de um termo abrangente, apesar da curta definição, a avaliação de dependabilidade costuma se inter-relacionar ao estudo de um conjunto de **Ameaças** que impedem

a entrega do serviço de maneira desejada; de **Meios** que são os responsáveis por garantir os resultados dos **Atributos** que, por sua vez, nada mais são do que métricas relacionadas à entrega do serviço ofertado da já então mencionada maneira justa e confiável. Estas propriedades podem ser esquematizadas através de uma árvore, como na Figura 2.2

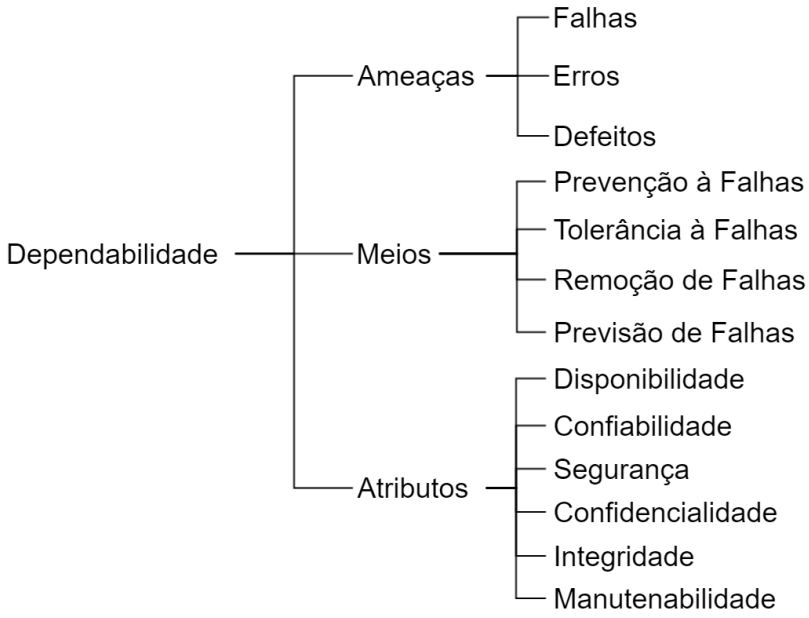


Figura 2.2: Árvore de Dependabilidade (baseado em AVIŽIENIS et al. (2001))

Os três principais ramos da árvore de dependabilidade (Ameaças, Meios e Atributos), são descritos nas subseções a seguir e as suas relações com esta dissertação são apontadas no decorrer do texto.

Ameaças à Dependabilidade

Existem três grandes ameaças à dependabilidade, as folhas da grande árvore englobam as falhas, erros e defeitos que impedem a entrega do serviço da maneira desejada.

Os erros são as ameaças inclusas no próprio *software* e, geralmente, lá estão por má programação no processo de desenvolvimento, ou estão ainda mais enraizados, tendo passado como despercebidos durante toda a etapa de planejamento do sistema; No geral, erros podem acarretar um estado de defeito, que por sua vez, refere-se ao momento em que um erro atinge a interface de serviço e altera seu funcionamento, tornando-se visíveis aos usuários; Enquanto que as falhas são ditas como as causas prováveis ou hipotéticas que levaram a ocorrência do erro AVIŽIENIS et al. (2001); REGALADO (2011).

Existem técnicas destinadas a mitigar o impacto das falhas nos sistemas computacionais, estas que estão no ramo vizinho ao das ameaças, e se intitulam Meios de Dependabilidade.

Meios de Dependabilidade

Para um sistema alcançar as métricas de dependabilidade e fornecer o serviço para qual foi programado, de forma justa e confiável, é necessária a combinação de quatro técnicas que visam a redução do impacto de falhas AVIŽIENIS et al. (2001): **Prevenção**, **Tolerância**, **Remoção** e **Previsão de Falhas**.

Prevenção a Falhas , o primeiro dos meios de dependabilidade é a prevenção às falhas, porém pode ser alcançada através da adoção de melhores técnicas para o controle de qualidade no processo de planejamento e fabricação tanto do *software* quanto do *hardware*, onde será executado AVIŽIENIS et al. (2001).

Exercer uma etapa de levantamento e avaliação de requisitos, bem como a adoção de programação estruturada para o processo de desenvolvimento e a aplicação de testes frequentes tende a ser o bastante para uma redução considerável da ocorrência de falhas no futuro.

Mas na ocorrência de falhas, torna-se imprescindível que algum mecanismo garanta que o sistema não fique indisponível aos seus usuários e continue entregando aquilo para que foi programado de maneira correta. Tais mecanismos são explicados a seguir.

Tolerância às Falhas consiste na entrega do serviço de maneira correta, mesmo na presença de falhas, e na capacidade que um sistema tem de se recuperar logo após a detecção de um erro AVIŽIENIS et al. (2001). Um sistema capaz de detectar a ocorrência de erros de forma automática e o local de acontecimento deste evento terá uma maior probabilidade de ser reparado antes da ocorrência de falhas catastróficas.

Depois de ser reparado, é ideal a todo e qualquer sistema a remoção da falha que reduziu a sua capacidade de operação, esta folha da árvore de dependabilidade é explicada logo abaixo.

Remoção de Falhas é um processo constante, sendo realizado durante o desenvolvimento de um sistema e no decorrer de todo o seu tempo de vida útil AVIŽIENIS et al. (2001). Quanto mais tempo dedicarmos às análises primárias e a etapa de validação, menor tempo tende a ser gasto removendo falhas no futuro, o que pode acarretar diretamente em um aumento na disponibilidade do produto ou serviço.

Se pudermos prever a ocorrência de falhas com o máximo possível de exatidão poderemos removê-las antes que ocorram, mitigando seu impacto sobre o fornecimento do serviço. Este que é o meio de dependabilidade explicado a seguir.

Previsão de Falhas ao se utilizar de técnicas para a análise comportamental de um sistema é possível prever, com certo índice de acurácia e confiança, quando o serviço poderá deixar de ser ofertado corretamente AVIŽIENIS et al. (2001). Com valores estimados é

possível reduzir e mitigar os impactos da ocorrência de uma falha no sistema, e através da aplicação de manutenções corretivas e preventivas é possível aumentar a continuidade no fornecimento do serviço e torná-lo mais confiável.

Confiabilidade esta, que é um dentre os seis **atributos de dependabilidade** apresentados na próxima Subseção.

Atributos de Dependabilidade

Os atributos de dependabilidade referem-se às metas a serem alcançadas por sistemas que almejam ser considerados justos e confiáveis. Um destes atributos, a disponibilidade, recebe um destaque maior em sua explanação, por tratar-se do foco desta dissertação de mestrado.

Disponibilidade , este termo pode ser resumido como a probabilidade de um sistema estar operacional, ou seja, esteja acessível aos seus usuários dentro de um período de tempo pré-estabelecido IEEE (1990). Esta probabilidade pode ser dada através do tempo em que o sistema tende a ficar disponível ou *Uptime* e do seu período de indisponibilidade *Downtime*, estes valores são legalmente determinados no SLA, sendo assim, a disponibilidade é uma métrica de grande importância para provedoras de serviços e infraestruturas. A Equação 2.1 representa o cálculo de disponibilidade MACIEL et al. (2011) para um sistema ou serviço.

$$A = \frac{E[Uptime]}{E[Uptime] + E[Downtime]} \tag{2.1}$$

onde **A** equivale à *availability* ou disponibilidade e **E** ao valor esperado.

Outra alternativa para a representação de disponibilidade de um sistema é em termos de tempo médio entre as falhas, Mean Time to Failure (MTTF) MACIEL et al. (2011), e do tempo médio entre os reparos, Mean Time to Repair (MTTR) MACIEL et al. (2011), como pode ser visto na Equação 2.2 MACIEL et al. (2011).

$$A = \frac{MTTF}{MTTF + MTTR} \tag{2.2}$$

Já a Equação 2.3 apresenta a disponibilidade em **número de noves** (NN) MACIEL et al. (2011), e quanto maior a quantidade, maior será a sua disponibilidade, um serviço com 99,44% de disponibilidade, também pode ser interpretado como um sistema com 2,25 noves de disponibilidade, por exemplo.

$$NN = \log_{10}(UA) \tag{2.3}$$

Mas, eis que surge a seguinte pergunta: Se um sistema possuir 100% de disponibilidade, como representar isto em números de nove? É praticamente impossível a garantia desse

valor, nem mesmo as maiores empresas que oferecem infraestruturas e serviços pela Internet anunciam uma disponibilidade tão alta. Em contrapartida, existe o cálculo da indisponibilidade do sistema, representado na equação anterior como **UA** MACIEL et al. (2011), e que pode ser calculado como o inverso da disponibilidade ou $1-A$.

A Equação 2.4 MACIEL et al. (2011) representa o cálculo para tempo médio entre as falhas do sistema.

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

Já o cálculo para se determinar o tempo médio entre os reparos do sistema depende diretamente do resultado proveniente da derivada anterior, e é representado pela Equação 2.5 MACIEL et al. (2011):

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

O período de *downtime* do sistema pode ser entendido como o período em que o serviço esteve indisponível dentro de um intervalo de tempo. O cálculo desta métrica leva em consideração a relação entre o tempo levado para uma equipe de manutenção locomover-se ao local da falha e a detecção de sua ocorrência, período é conhecido como *período-sem-reparo* ou *non-repair time* (NTR) e o tempo necessário para reparo da falha (TTR), deste modo podemos calcular o *downtime* de um serviço através da expressão $Downtime = NRT + TTR$ MACIEL et al. (2011).

Há um outro atributo de dependabilidade ligado diretamente aos tempos de falha do sistema, mas que diferentemente da disponibilidade, não leva em consideração os valores para reparo, esta é a **confiabilidade**, à seguir apresentada.

Confiabilidade é a probabilidade do sistema ter executado sua função até um limite de tempo previamente estabelecido e sem interrupções AVIŽIENIS et al. (2001); BOLCH et al. (2006).

A confiabilidade de um sistema pode ser medida de maneira semelhante ao cálculo da disponibilidade, porém as taxas relacionadas ao reparo do mesmo não são consideradas, a Equação 2.6 apresenta este cálculo.

$$R(t) = P(T > t), t \geq 0 \tag{2.6}$$

Onde T é a variável aleatória que representa o tempo para falha, e que está dentro do intervalo $[0,t]$ e R a confiabilidade KUO; ZUO (2003), do inglês *reliability*.

Segurança é uma métrica diretamente relacionada à confiabilidade do sistema, já que depende da inexistência de interrupções em seu fornecimento. Pode-se dizer que um sistema é

seguro quando a ocorrência de falhas catastróficas é pouco provável, porém não é possível dizer o mesmo sobre a existência de perigos eminentes AVIŽIENIS et al. (2001).

Integridade , um sistema íntegro é aquele onde alterações impróprias não serão aplicadas, sendo assim, podemos afirmar que a integridade de um sistema nada mais é do que um pré-requisito para as métricas de disponibilidade, confiabilidade e segurança AVIŽIENIS et al. (2001).

Manutenabilidade é a habilidade que um sistema tem de receber reparos e manutenções após a ocorrência de falhas AVIŽIENIS et al. (2001), ou a probabilidade de um sistema ser reparado após a ocorrência de uma falha em algum momento DANTAS (2013).

Confidencialidade , um sistema é dito confiável quando não há divulgação de caráter não autorizado de dados e informações sobre o mesmo AVIŽIENIS et al. (2001), mantendo aquilo que deve estar e continuar em segurança, sigiloso.

Disponibilidade Orientada à Capacidade (COA)

A disponibilidade orientada à capacidade avalia o quanto do sistema é realmente entregue a seus usuários e, para tanto, não considera apenas estados de atividade ou inatividade do sistema, mas sim o impacto real desses fatores na entrega do serviço. Em um sistema com dois servidores em operação paralela, como visto na Subseção 2.4.1, o serviço estará disponível, se pelo menos um dos servidores estiver operacional. Desta forma as probabilidades possíveis são: O serviço está sendo completamente entregue (os dois servidores estão operantes), serviço está sendo entregue pela metade (um único servidor está operacional) ou indisponibilidade total do serviço (todos os servidores estão em estado de falha ou sendo reparados). Deste modo, podemos calcular a disponibilidade orientada à capacidade pela Equação 2.7 MACIEL et al. (2011):

while π_i describes the steady state availability for the s_i state, a set with all available states know as *US* it is also considered and the maximum processing capacity of the system *MPC*. Thus, we can calculate the capacity-oriented availability by Equation 2.7 MACIEL et al. (2011):

$$COA = \frac{\sum_{s_{i \in US}} pc_i \times \pi_i}{MPC} \tag{2.7}$$

onde π_i descreve a disponibilidade estacionária do estado s_i pertencente ao conjunto de todos os estados de disponibilidade *US*, e pc_i é a capacidade de processamento operacional do estado s_i, já *MPC* descreve a capacidade máxima de processamento do sistema.

Existem técnicas destinadas à avaliação dos atributos de dependabilidade, bem como da disponibilidade orientada à capacidade, dentre elas estão os **modelos estocásticos**, explanados a seguir.

Modelos Estocásticos

São técnicas para modelagem e avaliação de sistemas computacionais, podendo, também, serem associadas a atributos de dependabilidade como disponibilidade e confiabilidade. Estes modelos são classificadas em combinatoriais (ou sem espaço de estados) e modelos com espaço de estados MALHOTRA; TRIVEDI (1994). Modelos combinatoriais capturam as condições que levam um sistema a falhar ou permanecer em funcionamento em termos de relacionamentos estruturais entre seus componentes MACIEL; LINS; CUNHA (1996). Já os modelos com espaço de estado representam o comportamento do sistema com base na ocorrência de eventos que levem a execução das rotinas de falha e reparo, que podem ser representadas através de taxas ou funções de distribuição MALHOTRA; TRIVEDI (1994). Modelos baseados em estado permitem uma representação de sistemas mais complexos do que modelos combinatoriais, todavia requerem um maior poder computacional para serem avaliados. As técnicas baseadas em espaço de estado utilizadas neste trabalho são apresentadas abaixo, elas diferem uma da outra em nível de abstração e no poder de representatividade de cada uma MALHOTRA; TRIVEDI (1994).

Diagramas de Bloco de Confiabilidade - RBD

RBD é um tipo de modelo proposto inicialmente para a análise de confiabilidade de sistemas e suas interações, esta técnica permite a utilização de combinações para modelar o comportamento de um sistema e seus módulos, produzindo uma análise confiável e que mais tarde teve suas características estendidas ao cálculo da disponibilidade dos sistemas computacionais, quer sejam estes pequenos e simples ou grandes e complexos MACIEL et al. (2011); MACIEL; LINS; CUNHA (1996).

Através de RBD é possível descrever a interação entre os diversos componentes do sistema, usando-se apenas de blocos de confiabilidade, cada bloco pode representar um único componente do sistema ou subcomponentes que compõe um subsistema. Em um modelo do tipo RBD, um sistema apenas estará disponível se for possível realizar o tracejo de uma linha, que se estende do vértice de origem (**Início**) até o seu vértice final (**Fim**), onde todos os componentes dentro desta linha tracejada estejam em estado de disponibilidade, ou seja, funcionais.

Um possível exemplo típico do funcionamento de um RBD está na Figura 2.3, esta figura representa um sistema com quatro componentes interconectados, sendo observados os dois principais tipos de blocos utilizáveis dentro deste tipo de modelo: Em série, como os Componentes 1 e 4, e em Paralelo como os Componentes 2 e 3. Este sistema estará disponível, como já mencionado, apenas se a linha entre início e fim for possível de ser traçada em, pelo menos, um dos possíveis caminhos.

Para delinear os caminhos possíveis dentro de um RBD, é necessário estabelecermos e determinarmos uma função estrutural capaz de avaliar a disponibilidade do sistema com base nos seus modos de operação. Consideremos um sistema S com dois componentes: 1 e 2, podendo

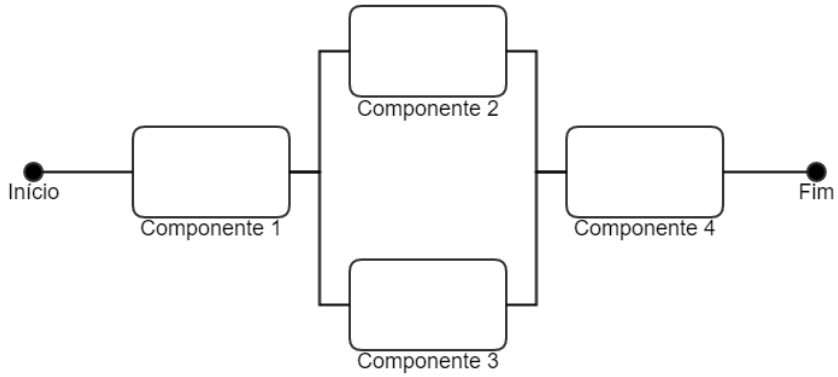


Figura 2.3: Exemplo de Diagrama de Bloco de Confiabilidade

este sistema estar em um estado de falha ou operacional, de acordo com o estado de cada um de seus componentes. O estado de um $componente_i$ é dado por uma variável aleatória x_i, tal que:

$$x = \begin{cases} 0, & \text{se o componente estiver em falha} \\ 1, & \text{se o componente estiver operacional} \end{cases} \tag{2.8}$$

A confiabilidade e a disponibilidade para nosso sistema caso os componentes estejam conectados em série podem ser calculadas através da Expressão 2.9 MACIEL et al. (2011).

$$R_S(t) = R_1(t) \times R_2(t) \tag{2.9}$$

Onde $R_1(t)$ é a confiabilidade do componente 1, e $R_2(t)$ é a confiabilidade do componente 2. Já para um sistema em série com n-elementos a Equação 2.10 MACIEL et al. (2011) é adotada.

$$R_S(t) = \prod_{i=1}^{n} R_i(t) \tag{2.10}$$

Para uma sequência de blocos em paralelo a Equação 2.11 pode ser utilizada MACIEL et al. (2011), com n representando o número em paralelo.

$$R_P(t) = 1 - \prod_{i=1}^{n} (1 - R_i(t)) \tag{2.11}$$

Uma outra estrutura típica de modelos RBD é a *k-out-of-n*, utilizada geralmente quando há um conjunto de componentes de um mesmo tipo distribuídos paralelamente, onde uma quantidade k de componentes pode entrar em estado falha e ainda assim o sistema permanecerá em funcionamento, como pode ser visto na Equação 2.12 que calcula sua confiabilidade TRIVEDI et al. (1996); MACIEL et al. (2011).

$$R_{k-out-of-n}(t) = \sum_{i-k}^{n} P(X = i) \tag{2.12}$$

É graças a esta visão de alto nível, que modelos do tipo RBD são largamente utilizados

na representação de sistemas, porém, com eles é muito difícil demonstrar a existência de pré-requisitos de funcionamento entre componentes dentro de um sistema, ou seja, é possível apenas representar módulos independentes, deste modo, para a representação de sistemas considerados mais complexos são necessárias técnicas mais refinadas de modelagem, como é o caso das SPN e CTMC.

Cadeias de Markov de Tempo Contínuo - CTMC

Cadeias de Markov são técnicas destinadas à modelagem estocástica. São capazes de descrever o funcionamento do sistema com base em um conjunto de estados e transições, estas que podem descrever eventos de falha e reparo de um sistema ou componente.

As cadeias de Markov são mais convenientes que os RBDs quando queremos descrever propriedades dinâmicas dos sistemas BOLCH et al. (2006). Cada transição em uma cadeia de Markov representa um processo estocástico $X(t)$, onde $t \in T$ é um conjunto de variáveis aleatórias definidas sobre o mesmo espaço de probabilidades, e capazes de assumir valores no espaço de estados $S_i \in S$ CASSANDRAS; LAFORTUNE (1999). Deste modo, se o conjunto T qual a variável pertence for discreto, ou seja, com $t = 1,2,3,...$, o processo é dito processo de parâmetro discreto ou tempo discreto, já se T for um conjunto contínuo, tem-se um processo de parâmetro contínuo ou tempo contínuo, assumindo assim distribuições geométricas (*Discrete-Time Markov Chain* (DTMC)) ou exponenciais (*Continuous-Time Markov Chain*(CTMC)) respectivamente SOUSA et al. (2009),MACIEL et al. (2011).

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 $X(t_{n+1})$ depende somente do último valor anterior $X(t_n)$ e não dos valores que o antecedem $X(t_0), X(t_1), ..., X(t_{n-1})$, isto é, para qualquer número real $X_0, X_1, X_2, ..., X_n, X_{n+1}$, $P(X_{n+1} = s_{n+1} \mid X_n = s_n, X_{n-1} = s_{n-1}, ..., X_0 = s_0 = P(X_{n+1} = s_{n+1} \mid X_n = s_n$ BOLCH et al. (2006). Esta característica é também intitulada como ausência de memória, e é comumente encontrada em distribuições do mesmo tipo ou que se assemelham as distribuições exponenciais ou geométricas. A ausência de memória garante que os valores do estado atual em uma cadeia de Markov são totalmente independentes dos valores dos estados anteriores GARG et al. (1995).

Através de uma cadeia de Markov é possível realizar a modelagem comportamental de um sistema graficamente utilizando apenas de estados e transições, a Figura 2.4 mostra um exemplo de uma cadeia de Markov com apenas dois estados: **UP** e **DOWN**, operante e não-operante, de um sistema genérico; e dois arcos: **Falha** e **Reparo**.

Além da representação gráfica similar à de autômatos finitos as cadeias de Markov podem ser representadas em forma de uma matriz, intitulada matriz de taxa de **transição Q**. Nesta matriz, estão contidos os detalhes e informações que descrevem as transições de todos os estados dentro de uma cadeia de Markov e que são utilizadas durante o processo de resolução. Os elementos localizados fora da diagonal principal representam a taxa de ocorrência dos eventos

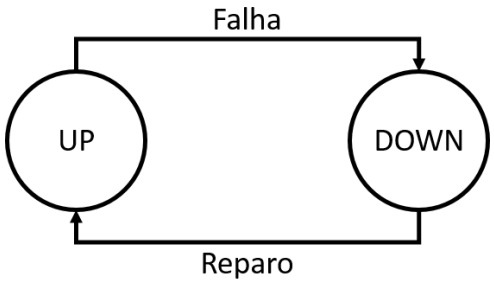


Figura 2.4: Exemplo de uma cadeia de Markov

que ocasionam na transição dos estados da cadeia de Markov. Já os elementos que se encontram na diagonal principal representam os valores necessários para que a soma entre elementos de cada linha seja igual à zero, tal que:

$$Q = \begin{pmatrix} q_{ii} & q_{ij} \\ q_{ji} & q_{jj} \end{pmatrix} \tag{2.13}$$

A probabilidade de cada estado fornece uma solução de regime estacionário com base em $\pi Q = 0$, onde Q é a matriz de estados e π é o autovetor correspondente ao autovalor unitário da matriz de transição, resultando em um vetor nulo. É importante ressaltar que a soma dos elementos do vetor de probabilidade π deve ser sempre igual a 1, ou seja, $\|\pi\| = 1$ MACIEL et al. (2011). Já as probabilidades de transição de estados são calculadas através da Equação 2.14.

$$p_{i,j}(s,t) = PX(t) = j \mid X(s) = i \tag{2.14}$$

Quanto a soluções que dependem do tempo, ou seja, do tipo transiente, tempos de execução longos são considerados, podendo-se mostrar que a probabilidade dos estados converge para valores constantes HERZOG (2002). O comportamento transiente da cadeia de Markov nos fornece informações de desempenho e dependabilidade sobre os instantes iniciais do sistema MACIEL et al. (2011).

É graças a essas características que as cadeias de Markov possuem um poder de representação maior que o dos RBDs, permitindo-as a possibilidade de modelar prioridades e dependências entre os componentes de um sistema, principalmente pelo fato de que as transições de uma cadeia de Markov podem assumir como valores probabilidades, taxas e funções de distribuição MACIEL et al. (2011).

Sistemas mais complexos com grande quantidade de estados, por vezes, tem sua avaliação de forma gráfica ainda mais complexa, fazendo necessária a utilização de abordagens que possam algum tipo de automação em processos e que possuam o mesmo poder de representação, estas são as SPNs, explicadas a seguir.

Redes de Petri Estocásticas - SPN

Redes de Petri são uma poderosa ferramenta para análise e modelagem comportamental de sistemas AGERWALA (1979); BALBO (2001). E foram propostas pela primeira vez em 1962 por Carl Adam Petri em sua tese de doutorado intitulada *Kommunikation mit Automaten* (Comunicação com Autômatos) MURATA (1989).

Os principais componentes das redes de Petri são os lugares (Figura 2.5a), transições (Figura 2.5b), arcos (Figura 2.5c) e marcações (Figura 2.5d). Os lugares são os possíveis estados alcançáveis pelo sistema, as transições são as ações por ele realizadas, já os arcos conectam os lugares as transições enquanto que as marcações representam a quantidade de recursos ou o estado atual do sistema MACIEL; LINS; CUNHA (1996).

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

Figura 2.5: Principais componentes de uma rede de Petri

A Figura 2.6a apresenta um exemplo de uma rede de Petri simples em seu estado inicial, antes do disparo de uma transição, os disparos de transições ou a ocorrência de ações em uma rede de Petri apenas são possíveis se o número de recursos/marcações for suficiente no lugar que as antecede e, para tanto, o peso dos arcos não pode ser maior que o número de marcações naquele lugar, o que pode ser visto através da Figura 2.6b que mostra a ocorrência do disparo.

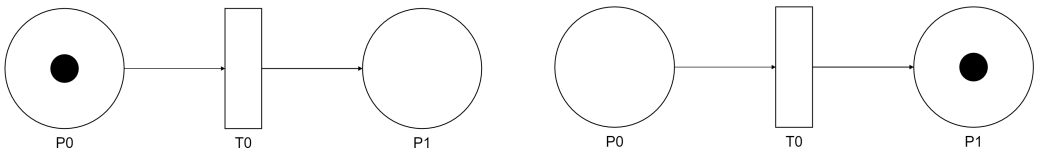
De acordo com MURATA (1989) uma rede de Petri pode ser definida através de uma $5-tupla$, do tipo $PN = (P,T,F,W,M_0)$, onde:

$P = (p_1, p_2, ..., p_n)$ é um conjunto finito de lugares,

$T = (t_1, t_2, ..., t_m)$ é um conjunto finito de transições,

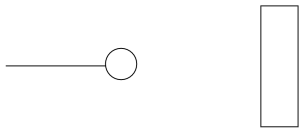
$F \subseteq (P \times T) \cup (T \times P)$ é um conjunto de arcos,

$W : F \rightarrow \{1, 2, 3, ...\}$ é a função de peso,



(a) SPN antes do disparo de transição **(b)** SPN após o disparo da transição

Figura 2.6: Exemplo de uma rede de Petri



(a) Arco Inibidor **(b)** Transição Temporizada

Figura 2.7: Componentes adicionais para uma rede de Petri estocástica

$M_0 : P \to \{0,1,2,3,...\}$ são as marcações iniciais,

$P \cap T = \oslash$ e $P \cap T \neq \oslash$. ...

O avanço no estudo das redes de Petri proporcionou uma evolução em seu poder de representação, e hoje graças às SPN é possível descrever sistemas assíncronos, temporizados, concorrentes e não-determinísticos GERMAN (2000). As transições em uma SPN são temporizadas e sua representação gráfica difere daquelas anteriormente apresentadas, como mostra a Figura 2.7b, e um novo tipo de arco intitulado de arco inibidor tem sua representação gráfica apresentada pela Figura 2.7a.

Arcos inibidores são capazes de determinar se um dado local possui ou não marcações, já as transições temporizadas possuem o seu tempo de disparo pré-estabelecido e baseado em uma distribuição de probabilidade. Já as transições anteriormente apresentadas passam a se chamar transições imediatas e possuem prioridade de disparo ante os demais tipos de transição.

A rede de Petri estocástica pode ser definida através de uma $9-tupla$, do tipo $SPN = (P,T,I,O,H,\Pi,G,M_0,ATTs)$, onde:

$P = (p_1, p_2, ..., p_n)$ é o conjunto de lugares,

$T = (t_1, t_2, ..., t_m)$ é o conjunto de transições imediatas e temporizadas $P \cap T \neq \oslash$,

$I \in (\mathbb{N}^n \to \mathbb{N})^{n \times m}$ é a matriz que representa os arcos de entrada (que podem ser dependentes de marcações),

$O \in (\mathbb{N}^n \to \mathbb{N})^{n \times m}$ é a matriz que representa os arcos de saída (que podem ser dependentes de marcações),

$H \in (\mathbb{N}^n \to \mathbb{N})^{n \times m}$ é a matriz que representa os arcos inibidores (que podem ser dependentes de marcações),

$\Pi \in \mathbb{N}$ é um vetor que associa o nível de prioridade de cada transição,

$G \in (\mathbb{N} \to \{verdadeiro, falso\})^m$ é o vetor que associa uma transição de guarda relacionada à marcação do lugar a cada transição,

$M_0 = P \to \{0,1,2,3,...\}$ é o vetor que associa a marcação inicial de cada lugar;

ATTs compreende o conjunto de atributos associados à transições, como a função de distribuição, política de memória, grau de ocorrência e o peso de cada transição.

É importante salientar que as transições temporizadas se caracterizam por semânticas distintas de disparos. Estas que podem ser de três tipos: *single server*, onde apenas uma marcação é disparada por vez, *multiple server* é possível realizar k disparos de uma única vez, e *infinite server* onde é possível realizar um número infinito de disparos MARSAN et al. (1994).

Redes de Petri estocásticas marcadas com um número finito de lugares e transições são isomórficas às cadeias de Markov MOLLOY (1981), deste modo, é possível gerar uma cadeia de Markov a partir de uma SPN que corresponda a estas características.

Em sistemas mais complexos, modelados através de cadeias de Markov de tempo contínuo, analisar quais componentes possuem maior impacto na infraestrutura ou serviço ofertado, tornam-se uma tarefa árdua, em virtude da quantidade de estados que podem ser assumidos. Para auxiliar este processo algumas técnicas foram desenvolvidas, uma delas é apresentada a seguir.

Análise de Sensibilidade

São métodos utilizados para se determinar quais componentes tem maior influência sobre uma métrica de interesse em um determinado sistema FRANK (1978). Com os resultados de uma análise de sensibilidade é possível definir onde se pode ou deve aplicar técnicas que possibilitem uma melhora nos resultados daquela métrica, como, por exemplo, a redução do tempo de reparo de um componente importante tende a aumentar a disponibilidade do sistema.

Existe uma série de técnicas utilizadas para se fazer uma análise de sensibilidade, dentre elas estão: **Design Of Experiments** (DOE) e **Percentage Difference**, explanadas a seguir.

Design of Experiments é uma técnica que permite a especificação de um número razoável de experimentos ou avaliações a serem realizadas dentro do sistema ou modelo estocástico que o representa, com base em um número de fatores (parâmetros) envolvidos e níveis (valores) por eles alcançados dentro do sistema. Definindo uma quantidade de parâmetros variáveis a serem alterados a cada nova etapa do experimento, por exemplo, valores de MTTF e MTTR do sistema operacional, podemos obter o máximo de informações possível sobre este sistema JAIN (1991); MONTGOMERY (2006).

Um prático exemplo de DoE é o Full Factorial, que consiste na realização de todas as possíveis combinações entre fatores e níveis dentro do sistema JAIN (1991). Isto nos permite encontrar os efeitos e impactos de todos os fatores e da interação entre eles na métrica de interesse. A Equação 2.15 apresenta como é realizada esta análise.

$$n = \prod_{i=1}^{k} n_i \tag{2.15}$$

onde k fatores, com o *i-ésimo* fator possuindo n_i níveis, deverá executar uma quantidade n de experimentos. Apesar de gerar todos os possíveis resultados uma desvantagem emerge com base

na quantidade de níveis para cada fator, tornando-o uma tarefa que demanda de grandes recursos computacionais de tempo. Uma recomendação para reduzir esta desvantagem é a redução do número de níveis para cada fator, esta que pode ser feita através de uma análise minuciosa das métricas de interesse JAIN (1991).

A diferença percentual foi a técnica para análise de sensibilidade escolhida para ser utilizada neste trabalho, em virtude de sua praticidade para implementação na linguagem de *scripts* da ferramenta Mercury SILVA et al. (2015). A diferença percentual consiste na fixação de uma lista de parâmetros, enquanto um parâmetro específico tem seu valor variado, este passo se repete até que todos os parâmetros tenham sido variados um a um. A Expressão 2.16 mostra como o cálculo da análise de sensibilidade com diferença percentual é feita.

$$SI = (\frac{D_{max} - D_{min}}{D_{max}}), \tag{2.16}$$

onde SI é o índice de sensibilidade para o parâmetro selecionado, D_{max} é o máximo valor que aquele parâmetro pode assumir e, D_{min} corresponde ao valor mínimo para aquele parâmetro.

Computação em Nuvem

Computação em nuvem é um paradigma que consiste na alocação dinâmica de recursos virtualizados de hardware e software com uma meta: prover todos os tipos de serviços através da Internet VAQUEIRO et al. (2009). Este paradigma provê uma integração entre modelos tecnológicos e diferentes tipos de serviço providos. Inclusos Infraestrutura como Serviço (IaaS), Plataforma como Serviço (PaaS), e aplicações sob demanda Software como Serviço (SaaS), SUN (2009), SA; SOARES; GOMES (2011). A Figura 2.8, apresenta uma relação entre os três principais tipos de nuvem quanto ao tipo de serviço prestado. Estes que são descritos em detalhes abaixo:

Infraestrutura como serviço (IaaS): Oferece versões virtuais de dispositivos de hardware para processamento ou armazenamento de dados IBM (2012), a Infraestrutura como Serviço é o tipo de serviço de computação em nuvem que cresce mais rapidamente, oferece versões virtuais de dispositivos de hardware para processamento ou armazenamento de dados IBM (2012) e é a base para PaaS e SaaS;

Plataforma como Serviço (PaaS) a nuvem fornece aos desenvolvedores uma série de ferramentas e produtos hospedados pelo provedor em sua infraestrutura de hardware IBM (2012). Através deste serviço pessoas do mundo inteiro podem implementar seus próprios aplicativos e disponibilizá-los para clientes e empresas. Este tipo de nuvem torna-se vantajoso para os programadores de várias maneiras, pois poderão acessar seus projetos de qualquer lugar e por qualquer computador que possua conexão a Rede Mundial de Computadores, e ao disponibilizar seus softwares no outro tipo de nuvem intitulado de SaaS, poderá levar seus

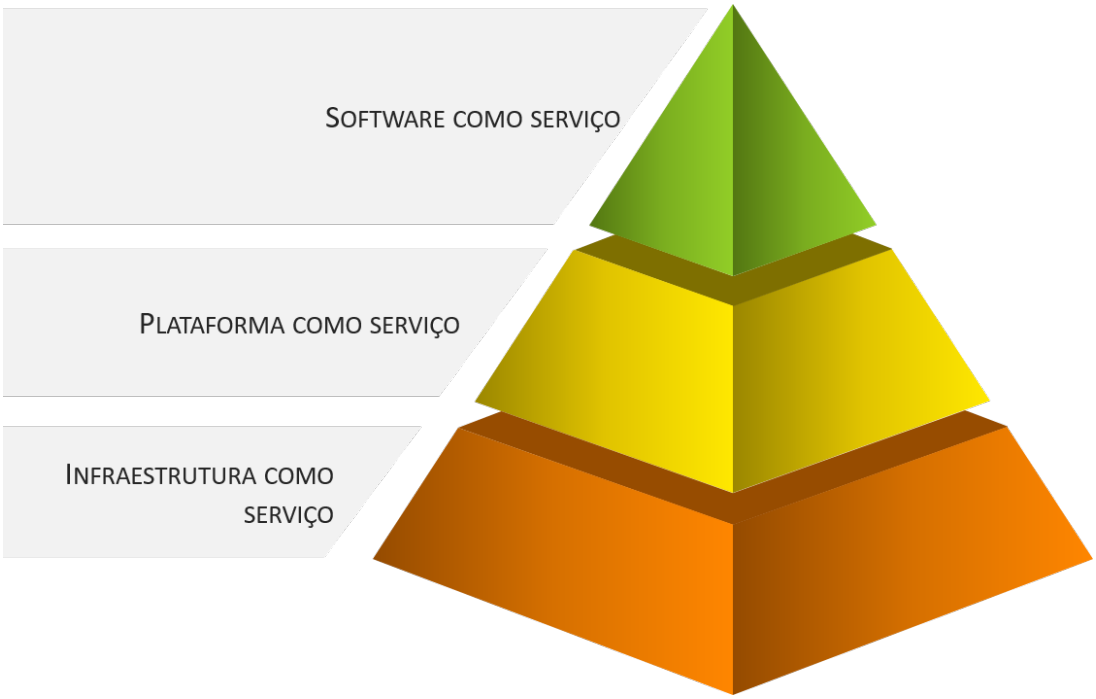


Figura 2.8: Relação entre os ambientes de computação em nuvem por tipo de serviço oferecido (baseado em RACKSPACE (2013))

produtos ainda em versão de desenvolvimento para seus clientes realizarem testes em tempo real podendo realizar sugestões de mudanças e alterações no projeto a qualquer momento;

Software como Serviço (SaaS) nele além do fornecimento da infraestrutura, a nuvem oferece softwares que possam ser acessados a partir de qualquer parte do mundo, grandes exemplos deste tipo de serviço são o Gmail e o YouTube da Google, o Outlook da Microsoft e demais serviços de e-mail como o Yahoo!, este é o tipo de nuvem mais largamente utilizado. E proporciona a seus usuários a possibilidade de armazenamento, visualização e reprodução de arquivos de vídeos, de imagens e de sons.

A computação em nuvem originalmente se divide em três vertentes relacionadas à forma em que o acesso aos serviços providos é disponibilizado, ou seja, sua natureza, a Figura 2.9 apresenta uma relação entre os tipos de nuvem pelo tipo de acesso. Estes três diferentes tipos de classificação são conhecidos simplesmente como os tipos Públicas, Privadas e Híbridas IBM (2012); NIRIX (2015).

Pública: Está disponível a todos na Internet. Qualquer usuário pode se inscrever para usar este tipo de nuvem sem nenhuma restrição, exceto a necessidade de uma conexão com a rede mundial de computadores IBM (2012);

Privada: É um ambiente onde os serviços são oferecidos a um número limitado de usuários e que requer alguma forma de autenticação. Geralmente são isoladas do meio externo através de barreiras, como um *firewall* IBM (2012), por exemplo;

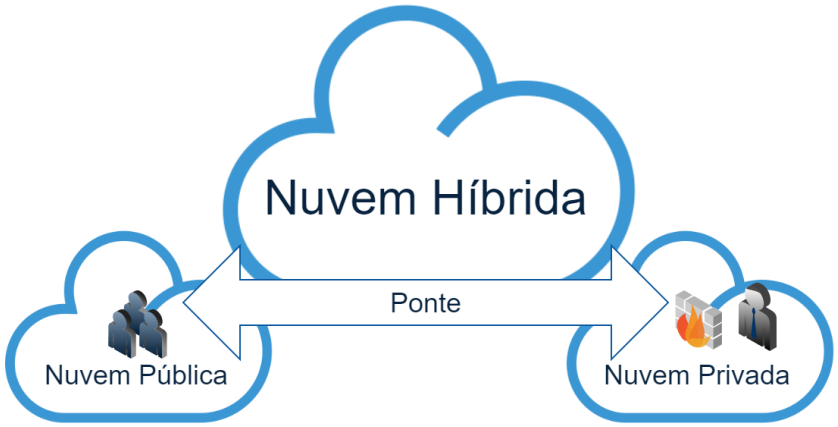


Figura 2.9: Relação entre os ambientes de computação em nuvem por tipo de acesso (Baseado em NIRIX (2015))

Híbrida: Através delas é possível fornecer serviços do tipo privado, utilizando os recursos provenientes de uma nuvem pública, para tanto, usualmente, limita-se o seu acesso através da utilização de redes do tipo Virtual Private Networks (VPN) IBM (2012).

A Subseção a seguir apresenta a plataforma de computação em nuvem utilizada no desenvolvimento desta pesquisa de mestrado.

Plataforma Eucalyptus

Elastic Utility Computing Architecture Linking Your Programs To Useful Systems ou simplesmente Eucalyptus JOHNSON et al. (2010) é uma plataforma de computação em nuvem aberta e renomada, capaz de prover a seus usuários a possibilidade de oferecer todos os tipos de serviço, bem como infraestruturas com acesso híbrido e privado JOHNSON et al. (2010). Os principais componentes desta plataforma e suas aplicações são listados abaixo:

Controlador de Nuvem (CLC): É o componente que permite à usuários o acesso à infraestrutura de nuvem através de aplicações que conectem-se à Internet, sendo também o responsável pela gerência e provimento de subserviços, como rede e armazenamento. Com o CLC é possível monitorar as instâncias em execução e definir quais clusters ou agrupamentos deverão realizar o provisionamento de novas instâncias;

Controlador de Cluster (CC): Controlador de Agrupamentos é o responsável por definir qual máquina física pertencente a um cluster dentro da infraestrutura de nuvem, que deverá instanciar uma máquina virtual. Ele verifica informações dos recursos disponíveis, possíveis horários para cada instância, e as requisições feitas pelo CLC;

Controlador de Armazenamento (SC): Controlador de armazenamento e responsável por prover o armazenamento em blocos para as máquinas virtuais, sendo capaz de criar e gerenciar

blocos de armazenamento e protegê-los de acesso indevido por outras máquinas em execução no nó;

Walrus: Armazenamento em arquivos, que permitem aos usuários o envio e coleta de informações e imagens de máquinas virtuais, através da Internet e seus protocolos possibilitam a migração e o download das máquinas virtuais;

Controlador de Nó (NC): Componente em execução nos nós físicos da infraestrutura. Conecta-se diretamente ao sistema operacional e ao *hypervisor* em execução neste computador, sendo responsável por manter o ciclo de vida das máquinas virtuais: Instanciação, monitoramento e término. Enviando os dados coletados por cada etapa do ciclo ao CC.

A interação entre cada um destes componentes pode ser vista através da Figura 2.10, que mostra o que acontece a partir do acesso à infraestrutura por meio de um computador pessoal, até os níveis mais baixos de ramificação dos *clusters* de uma arquitetura.

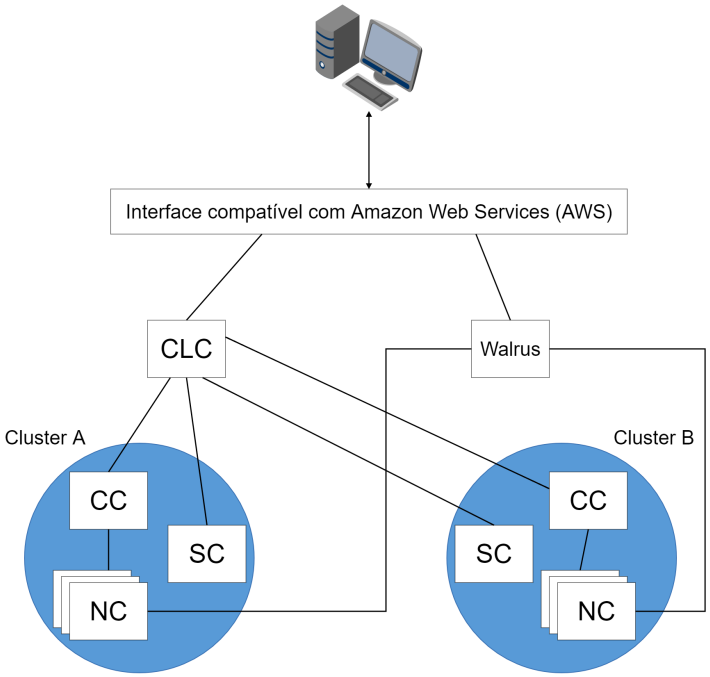


Figura 2.10: Relação entre os componentes da plataforma Eucalyptus (Baseado em JOHNSON et al. (2010))

Para saber o quanto do serviço realmente é entregue, diante da plataforma apresentada, partimos para a disponibilidade orientada à capacidade.

Para a criação dos modelos e avaliação de métricas apresentadas até aqui foram utilizadas ferramentas que facilitaram o processo de concepção. Neste C apítulo foram apresentadas as abordagens, técnicas e meios utilizados para a concepção desta pesquisa de mestrado, o Capítulo 3 apresenta, em detalhes, cada uma das arquiteturas propostas, bem como a metodologia utilizada para a criação destas arquiteturas.

3
Modelos

Neste Capítulo, são apresentadas a metodologia para modelagem e avaliação (Seção 3.1) e os modelos que integram esta dissertação (Seção 3.2). É a partir deles que podemos dar início à proposição de infraestruturas de sincronização de dados com a mais alta disponibilidade e o menor custo possível, e que possam atender as necessidades tanto das provedoras, quanto das contratantes e/ou usuários deste tipo de serviço.

Metodologia para Modelagem e Avaliação

Esta seção se inicia com a metodologia adotada por esta pesquisa para realização da avaliação de disponibilidade, custos de implantação, disponibilidade orientada à capacidade e confecção de modelos estocásticos. A sequência de passos que descreve as principais etapas envolvidas neste processo é representada pela Figura 3.1, e é transcrita na sequência.

Entendimento do sistema e definição das métricas de interesse, esta etapa caracteriza-se pela compreensão do sistema, identificação de seus componentes, aplicações e funcionalidades, objetivando uma delimitação no âmbito do trabalho a ser executado, como métricas de disponibilidade, consumo energético e avaliação de custos para implantação. A compreensão do sistema requer grande atenção e cuidados especiais por parte do avaliador, para assim, evitar erros de interpretação e comprometimento das demais etapas da metodologia de modelagem e avaliação do serviço de sincronização de dados. Compreendendo bem o serviço que queremos oferecer, seremos capazes de particioná-lo em subsistemas menores, além de identificar com maior facilidade erros de projeto e desenvolvimento do sistema;

Definição da Arquitetura Básica, esta etapa consistiu na elaboração de uma arquitetura contendo os componentes mínimos necessários ao funcionamento de um serviço de sincronização de dados hospedado em plataforma de computação em nuvem. Todas as arquiteturas variantes tomam a arquitetura básica como alicerce, já que os submodelos que representam o frontend e o nó da arquitetura geram valores de MTTF e MTTR utilizáveis a posteriori por cada uma delas;

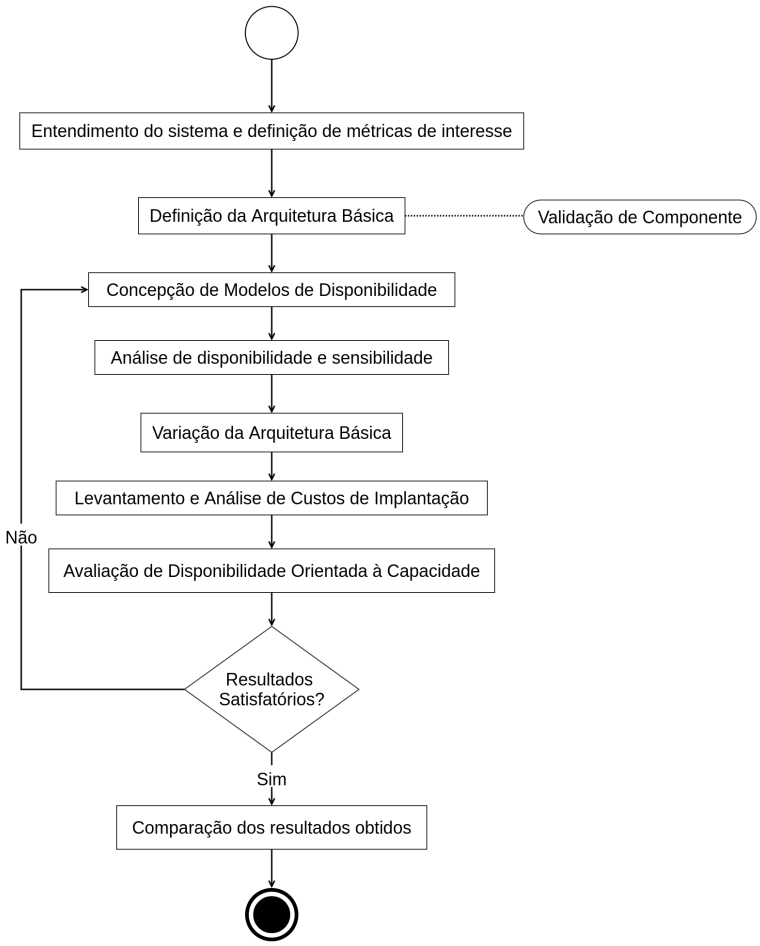


Figura 3.1: Metodologia Adotada

Validação de Componente é a etapa que ocorre paralelamente a definição da arquitetura básica, e caracteriza-se pela realização de experimentos de injeção de falhas e aplicação de reparos ao servidor de sincronização hospedado na plataforma Eucalyptus. É importante salientar a adoção de técnicas conhecidas como fatores de redução, que foram aplicadas a *scripts bash* utilizados para automatização e aceleração do processo de ocorrência de falhas e reparos do componente;

Concepção de Modelos de Disponibilidade, conhecendo o sistema e seus principais componentes somos capazes de prover os primeiros modelos baseados em estados e quais modelos serão utilizados para representar o comportamento da arquitetura básica, depois os variar de acordo com a nossa necessidade ou criação de novos cenários, buscando sempre a melhoria de métricas de interesse ou a melhor compreensão do comportamento do sistema avaliado. RBDs, CTMCs e SPNs devem representar de forma fiel o funcionamento do sistema e sua aplicabilidade ao mundo real, caso contrário tornam-se artefatos sem utilidade para as etapas posteriores;

Análise de disponibilidade e sensibilidade de posse dos modelos que representam o compor-

tamento da arquitetura básica, somos capazes de avaliar nossa métrica de interesse, ou seja, sua disponibilidade, para tanto, RBDs e CTMC são dispostos de maneira hierárquica, com a finalidade de encontrar um único valor de referência, logo após, a aplicação da técnica de diferença percentual aos componentes da arquitetura, propiciaram uma análise de sensibilidade através da variação dos valores de falha e reparo verificando qual ou quais dos componentes apresentam um maior impacto na disponibilidade do sistema;

Variação da Arquitetura Básica com base nos valores obtidos através da análise de sensibilidade, é possível utilizar de técnicas destinadas à melhoria do sistema e dos valores de disponibilidade associados. Um conjunto de técnicas utilizado em grande escala é caracterizado como redundância e, consiste na distribuição do mesmo sistema ou serviço em mais de um computador ou máquina virtual paralelamente, deste modo, em caso de falha do componente em uso o redundante passa a assumir o papel de fornecimento em um tempo menor do que o necessário para reparo do componente principal. Neste estudo uma série de arquiteturas redundantes baseadas na arquitetura básica são apresentadas, além da criação de novos modelos para representar o comportamento do serviço de sincronização de dados distribuído;

Levantamento e Análise de Custos de Implantação depois de estabelecermos os componentes necessários para o funcionamento de cada arquitetura, básica e suas variações, podemos avaliar valores de consumo energético para máquinas operantes e em *standby*, e os custos do kilowatt/h, em Recife, no mês de Abril de 2016, em seguida, uma avaliação dos valores para aquisição de cada máquina, dispositivo de rede e infraestrutura que integraria e garantiria o funcionamento de cada ambiente proposto;

Avaliação de Disponibilidade Orientada à Capacidade através dos modelos gerados também é possível determinar quanto do sistema está sendo devidamente entregue, para tanto a definição do que vem a ser a capacidade de nosso sistema precisa ser levantada e analisada, levando-se em conta as rotinas de falha e reparo, além do período de chaveamento entre um componente e outro, responsáveis pela entrega do serviço de forma justa e confiável;

Resultados Satisfatórios? se sim, ou seja, todas as variações da arquitetura básica que planejávamos construir foram devidamente geradas e avaliadas, nos permitindo passar à etapa seguinte, todavia, caso os resultados obtidos até o presente momento tenham sido insatisfatórios, ou seja, as arquiteturas variantes propostas não apresentaram uma melhoria considerável nas métricas analisadas, voltaremos à etapa de concepção de modelos e repetiremos os demais processos um a um;

Comparação dos resultados obtidos aqui os valores obtidos pela avaliação de cada arquitetura são comparados e, utilizando-se de técnicas para normalização de grandezas e de distância euclidiana, somos capazes de determinar relações que buscam prover o melhor custo

× benefício. A partir daqui, todos os artefatos (modelos) estarão construídos e prontos para serem entregues aos gestores e administradores de infraestruturas de computação em nuvem e, que planejam oferecer um serviço de sincronização de dados em seu ambiente.

Modelos

Nesta Seção, as arquiteturas de alto nível propostas são apresentadas, além dos modelos baseados em estado (SPN, CTMC e RBD) que representam o comportamento de cada um de seus principais componentes com base em rotinas de falha e reparo são mostrados. Os modelos são dispostos de maneira hierárquica através da linguagem de *scripts* da ferramenta Mercury (ver A), deste modo, modelos distintos como RBDs e CTMCs podem ser utilizados para avaliar um mesmo sistema, ou para a representação de um subsistema dentro de um modelo maior. É o que acontece a arquitetura básica, mostrada a seguir.

Arquitetura Básica

A arquitetura básica consiste de dois computadores, um frontend e um nó, esta que é a quantidade mínima de recursos necessários, quando se desejar ter uma máquina especifica para o gerenciamento dos recursos e disposição do nosso serviço na plataforma Eucalyptus. A Figura 3.2 apresenta uma visão de alto nível desta arquitetura, que tem como foco a área demarcada pelo quadrado pontilhado, ou seja, o lado da provedora de serviço, significando que tudo que ocorrer entre a rede e o lado do cliente não será contemplado por nossos modelos, incluso a rede e o dispositivo pessoal.

A definição de uma arquitetura básica torna-se importante, pois através dela somos capazes de compreender melhor o funcionamento do serviço que queremos oferecer e como os principais componentes da infraestrutura interagem entre si, o que nos possibilita a geração de modelos que representam os modos operacionais do serviço.

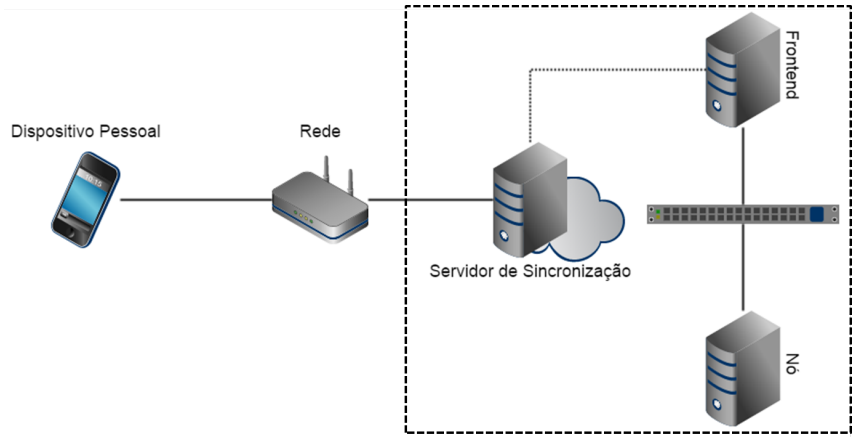


Figura 3.2: Visão de Alto nível da Arquitetura Básica

O serviço de sincronização de dados é configurado em uma máquina virtual gerenciada pelo controlador de nó em execução na máquina física nó, enquanto que a máquina frontend será responsável pela gerência de toda a infraestrutura de nuvem via controlador de nuvem.

Nesta arquitetura, o serviço apenas estará disponível em caso de disponibilidade de todos os componentes na máquina nó e frontend, uma falha no nó que hospeda a máquina virtual e que contém o serviço de sincronização tornará indisponível seu acesso, enquanto que uma falha no frontend acarretará na indisponibilidade total da plataforma.

O modo operacional que descreve o funcionamento do serviço de sincronização de dados na arquitetura proposta é obtido através da seguinte expressão lógica: $MO_{sistema} = Frontend \wedge No$. Para representação gráfica e comportamental deste sistema RBDs e CTMCs foram utilizados, representando a relação entre cada um dos componentes da plataforma de computação em nuvem.

Para modelagem do frontend e nó optamos por RBDs, em virtude da inexistência de prioridades entre os componentes do Eucalyptus; onde na falha de qualquer um deles todo o sistema estará comprometido e em estado de indisponibilidade. O primeiro RBD apresenta uma visão de alto nível do sistema, com dois blocos representando os subsistemas de frontend e nó, como mostra a Figura 3.3. Estes RBDs são baseados nos que foram propostos em DANTAS et al. (2012b, 2015).

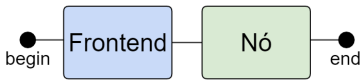


Figura 3.3: RBD: Arquitetura Básica

Cada bloco engloba um MTTF e MTTR do seu respectivo componente, permitindo assim o cálculo de sua disponibilidade, o que é feito através da seguinte expressão: $A = A_{Frontend} \times A_{No}$. Já o subsistema que representa o bloco frontend é representado na Figura 3.4. Este subsistema é composto pelos blocos: hardware (HW), sistema operacional (SO), e os seguintes componentes da plataforma Eucalyptus CLC, CC, SC e Walrus DANTAS et al. (2012b). O modo operacional que descreve o funcionamento deste submodelo é dado pela expressão: $MO_{frontend} = HW \wedge SO \wedge CC \wedge CLC \wedge SC \wedge Walrus$.

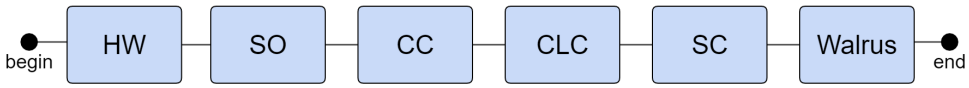


Figura 3.4: RBD: Subsistema do frontend

A expressão que determina o cálculo de disponibilidade deste subsistema é: $A_{Frontend} = A_{HW} \times A_{OS} \times A_{CC} \times A_{CLC} \times A_{SC} \times A_{Walrus}$. É importante salientar que assumimos uma distribuição exponencial para os valores para cada taxa de falha (λ) e reparo (μ) de cada bloco em nos modelos RBD apresentados e equivalem ao inverso do tempo $\frac{1}{\lambda}$ e $\frac{1}{\mu}$ respectivamente.

A Figura 3.5 apresenta o modelo RBD para o subsistema do nó. Este que é composto pelo hardware (HW), sistema operacional (SO), o hypervisor responsável pela gerência das

máquinas virtuais, controlador de nó (NC) e serviço de sincronização, que, por sua vez, engloba a máquina virtual.

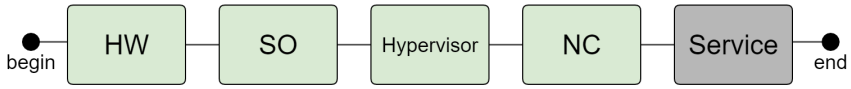


Figura 3.5: RBD: Subsistema do nó

A expressão lógica que representa o modo operacional do subsistema do nó é: $MO_{no} = HW \wedge SO \wedge Hypervisor \wedge NC \wedge Service$. O serviço recebe uma cor especial por tratar-se de um outro subsistema, representado por uma CTMC. A disponibilidade para o subsistema do nó pode ser obtida através da expressão: $A_{Node} = A_{HW} \times A_{SO} \times A_{Hypervisor} \times A_{NC} \times A_{\textbf{serviço}}$.

Diferentemente do nó e frontend que tiveram seus comportamentos modelados através de RBDs, o subsistema do serviço precisou ser modelado através de uma CTMC. A escolha se deve ao fato da proposição de um mecanismo de reparo baseado na reinicialização da máquina virtual a uma taxa *reboot*, visando à redução do período de indisponibilidade do serviço. Esta CTMC com nove estados é apresentada através da Figura 3.6.

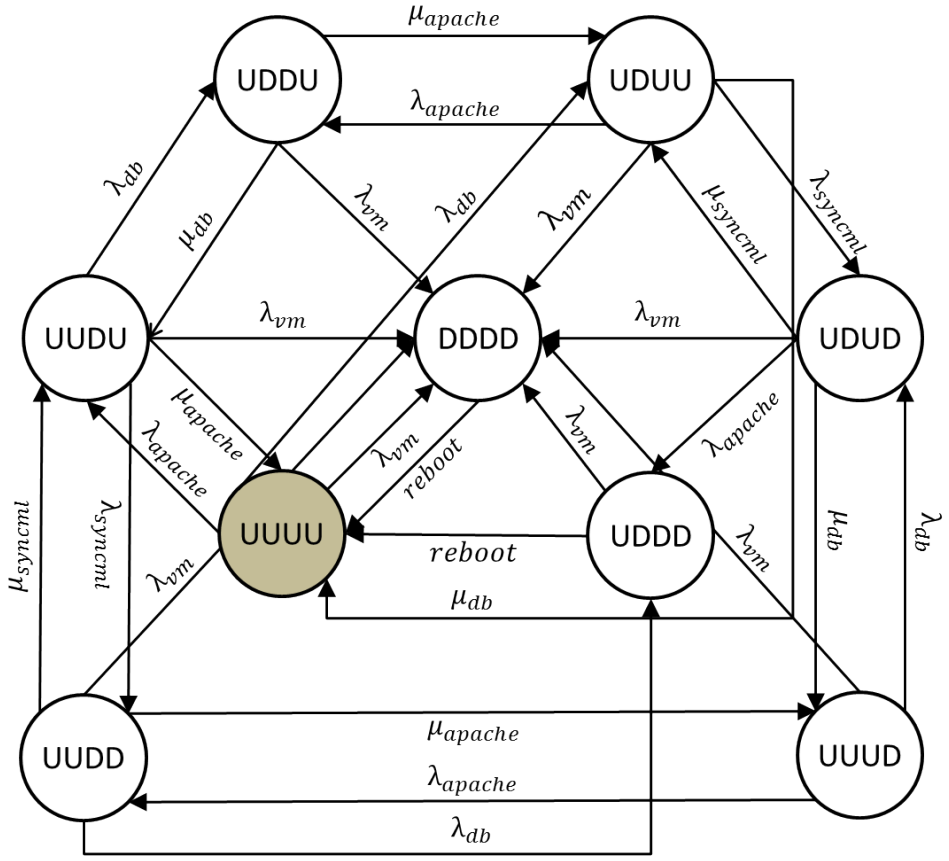


Figura 3.6: CTMC: Subsistema do Serviço

Para a CTMC não foi possível realizar a extração de uma expressão fechada para a disponibilidade do serviço, dada a sua complexidade, esta atividade não foi possível nem mesmo com a utilização de softwares como o Mathematica.

A notação adotada para definir cada um dos componentes nos estados da CTMC resume-se a dois caracteres, **U**, para quando o status deste componente for **disponível** e **D** caso esteja **indisponível**; O primeiro caractere representa a VM, o segundo o banco de dados, já o terceiro o Apache Web Server, enquanto que o último deles representa o status do servidor de sincronização. Os nove possíveis estados da CTMC são: UUUU, UUUD, UDUD, UUDU, UDUU, UUDD, UDDU, UDDD e DDDD. Nesta CTMC, o estado UUUU é o único onde o serviço está disponível e sendo provido como deveria. A Tabela 3.1 apresenta o resumo para cada estado da CTMC.

Tabela 3.1: Status do Serviço

Estado	Significado	Status
UUUU	VM UP, Apache UP, Database UP, SyncML UP	Disponível
UUUD	VM UP, Apache UP, Database UP, SyncML DOWN	Indisponível
UUDU	VM UP, Apache UP, Database DOWN, SyncML UP	Indisponível
UDUU	VM UP, Apache DOWN, Database UP, SyncML UP	Indisponível
UDUD	VM UP, Apache DOWN, Database UP, SyncML DOWN	Indisponível
UUDD	VM UP, Apache UP, Database DOWN, SyncML DOWN	Indisponível
UDDU	VM UP, Apache DOWN, Database DOWN, SyncML UP	Indisponível
UDDD	VM UP, Apache DOWN, Database DOWN, SyncML DOWN	Indisponível
DDDD	VM DOWN, Apache DOWN, Database DOWN, SyncML DOWN	Indisponível

A transição entre um estado e outro dentro de uma CTMC se dá através da ocorrência de falhas representadas pela taxa λ e reparos pela taxa μ, cada taxa de falha e reparo desta CTMC tem seu significado descrito na Tabela 3.2. Estes valores foram extraídos dos trabalhos realizados em DANTAS et al. (2012b,a); SOUSA et al. (2009); MELO et al. (2014); ESTEVAN COSTA; BRANCHER (2013).

Tabela 3.2: Taxas de Falha e Reparo do Serviço - CTMC

Taxa	Significado
λ_{syncml}	Falha do SyncML
μ_{syncml}	Reparo do SyncML
λ_{apache}	Falha do Apache
μ_{apache}	Reparo do Apache
λ_{db}	Falha do Banco de Dados
μ_{db}	Reparo do Banco de Dados
λ_{vm}	Falha da VM
reboot	Taxa para reparo da VM com os serviços

A disponibilidade do serviço na atual CTMC é calculada através da probabilidade do sistema estar no estado **UUUU**, este que é o único onde há provimento do serviço. Neste estado, a ocorrência de uma falha é possível em qualquer um dos componentes, caso ocorra no SyncML a uma taxa λ_{syncml}, chegaremos ao estado **UUUD**, caso ocorra no banco de dados a uma taxa λ_{db} chegaremos ao estado **UDUU**, no Apache a uma taxa λ_{apache} levando ao estado **UUDU**. Em último caso, na própria VM, esta falha pode ocorrer a partir de qualquer um dos outros estados da

CTMC à uma taxa λ_{vm} alcançando o estado **DDDD**. Reparos a partir de cada um desses estados de erro são possíveis a taxas μ_{syncml}, μ_{db}, μ_{apache} e *reboot* levarão o sistema de volta ao estado de disponibilidade.

Caso o sistema já esteja no estado de falha **UDUU**, onde houve falha no banco de dados, outras falhas podem ocorrer, tanto no Apache Web Server a uma λ_{apache} levando o sistema ao estado **UDDU**, quanto no SyncML a uma taxa λ_{syncml} chegando ao estado **UDUD**; reparos nesses estados levarão o sistema de volta ao estado UDUU.

Se no estado **UUDU** onde houve falha no Apache Web server, uma falha no banco de dados levará o sistema ao estado **UDDU** ou alcançar o estado **UUDD** através de uma falha no SyncML.

No estado **UUUD**, uma falha no banco de dados levará ao estado **UDUD**, ou através de falha no Apache web server acarretará no estado **UUDD**.

Em caso da ocorrência de falhas no SyncML e Apache Web Server, estaremos no estado **UUDD**, que precisará de uma sequência de dois reparos para tornar o sistema disponível novamente: μ_{syncml} e μ_{apache}, caso esses reparos não ocorram o estado **UDDD** pode ser atingido através de uma falha no banco de dados, e a partir daqui só poderemos alcançar o estado **DDDD** via falha na VM ou voltarmos ao estado **UUUU** por meio de reinicialização da máquina virtual.

Se no estado **UDUD** onde falhas já ocorreram no banco de dados e SyncML, a ida para o estado **UDDD** é novamente possível, mediante ocorrência de uma falha no Apache Web Server.

Validação do servidor de sincronização de dados hospedado arquitetura básica

A validação caracteriza-se como requisito primário para comprovação de viabilidade dos modelos apresentados, nesta dissertação, a validação ocorreu através da elaboração de *scripts* para monitoramento do comportamento do sistema, enquanto que uma rotina de injeção de falhas, também desenvolvida através de *scripts*, foi aplicada a um ambiente configurado para testes.

É importante salientar que neste trabalho validamos o servidor de sincronização, os valores para os demais componentes foram obtidos da literatura. Optamos pela aplicação dos métodos propostos por EJLALI et al. (2003) para provocar falhas e promover reparos de forma intencional desses dois componentes em nossa infraestrutura de testes, na tentativa de acelerar a ocorrência dos eventos de falha, em virtude da limitação temporal para a realização deste trabalho. Logo em seguida aplicamos os cálculos estatísticos propostos em KEESE (1965) buscando encontrar os valores obtidos pela avaliação dos nossos modelos dentro do intervalo de confiança contendo os valores de disponibilidade obtidos pela experimentação.

Em virtude da quantidade de tempo necessária para a ocorrência de falhas, técnicas conhecidas como "fatores de redução"ARAUJO et al. (2011) foram utilizadas em nossos *scripts* visando uma aceleração na ocorrência dessa rotina básica em sistemas computacionais, a Tabela 3.3 apresenta os valores utilizados por nossa rotina, considerando um fator de redução na casa de 100 para falha.

Para a infraestrutura de testes dois computadores foram alocados, um frontend e um nó,

Tabela 3.3: Fator de redução de MTTF

Componente	Falha Real	Falha com redução	Reparo
SyncML	788,4 h	7,88 h	1 h min

Tabela 3.4: Número de Ocorrências

Descrição	**Quantidade**
SyncML Falha	17
SyncML Reparos	17
Total	34

com a plataforma Eucalyptus como sistema operacional da nuvem, o serviço de sincronização utilizado foi o Funambol, um *framework* baseado na especificação do SyncML e de código aberto. A validação consistiu na consulta dos componentes necessários para o funcionamento do serviço, este que só estaria disponível se todos os componentes o estivessem. Os *scripts* utilizados tanto para monitoramento quanto para a injeção de falhas, encontram-se disponíveis no Apêndice F deste documento.

Após a especificação dos valores para e realização de experimentos foi possível a utilização da distribuição-F disponível na ferramenta Minitab, considerando um número de ocorrências de falha e reparo, estabelecemos um intervalo de confiança de 95%. A Tabela 3.4 apresenta o número de ocorrências investigado juntamente ao grau de liberdade, que trata-se da soma dessas ocorrências.

Ao todo 34 ocorrências foram registradas, das quais 17 representando estado de falha e outras 17 o reparo do servidor de sincronização, o experimento esteve em execução por aproximadamente 8 dias, e com base no intervalo de confiança e grau de liberdade obtidos pudemos determinar os intervalos máximo e mínimo que delimitam a distribuição-F, a Tabela 3.5 apresenta estes valores.

A relação entre os tempos para falha e reparo resultou em um número ρ, que precisou ser calculado através da Equação $\rho = \frac{F_n}{R_n}$, este valor nos permitirá mais a frente encontrar os valores mínimos e máximos representando o intervalo de confiança, considerando F_n o tempo total em estado de falha e R_n o tempo total em funcionamento do sistema.

De posse do valor de ρ e dos valores mínimos e máximos da distribuição-F, podemos calcular o valor mínimo (ρ_L) e máximo (ρ_U) de ρ, ao dividi-lo respectivamente pelos valores mostrados na Tabela 3.5 KEESE (1965).

Como último passo desta análise precisamos encontrar os já mencionados valores mí-

Tabela 3.5: Intervalos para distribuição-F

Descrição	**Valor**
Valor Mínimo	0,5948
Valor Máximo	1,981

Tabela 3.6: Intervalo de confiança para A e ρ

Intervalo de Confiança de 95%		
ρ	ρ_U	0,045292
	ρ_L	0,17774
A	A_U	0,956671
	A_L	0,849084

nimos e máximos para disponibilidade, A_L e A_U respectivamente, que fornecerão a validação do experimento se o valor encontrado pertencer ao intervalo de disponibilidade resultante das Expressões 3.1 e 3.2 para valores injetados.

$$A_L = \frac{1}{1+\rho_L} \tag{3.1}$$

$$A_U = \frac{1}{1+\rho_U} \tag{3.2}$$

A Tabela 3.6 resume os valores encontrados por nossas equações, e validam o modelo que apresenta uma confiança de 95% de que o valor 0,917664329 encontrado para a disponibilidade está dentro do intervalo que vai de A_U à A_L.

Após a validação da aplicação responsável pela sincronização, somos capazes de concluir que os valores apresentados pelos modelos são viáveis e se assemelham aos que seriam obtidos em uma arquitetura real. A próxima Seção apresenta estratégias de redundância utilizadas com o propósito de elevar os valores de disponibilidade e justificar o porque disso ocorrer.

Redundância no Nó

Três arquiteturas mais tarde analisadas são propostas, avaliando o impacto de três tipos de redundância nos nós: *Hot Standby, Warm Standby e Cold Standby*, todas consistem de três máquinas físicas, uma representando o frontend e duas os nós redundantes. A Figura 3.7 apresenta a visão de alto nível que representa a relação entre as máquinas. Vale salientar que na visão de alto nível o único componente com representação virtual é o servidor de sincronização.

Os mesmos componentes estão presentes em ambos os nós da arquitetura redundante, mas agora há variação no modo operacional do sistema, a expressão lógica que a representa é: $MO_{servico} = frontend \wedge (no1 \vee no2)$. Pelo menos o frontend e um dos nós precisa estar operante para que o serviço continue a ser provido e atenda a seus usuários.

Os modelos redundantes são variantes da arquitetura básica, e fazem uso dos valores por ela providos, além de possibilitar o reuso de alguns dos modelos apresentados mas sempre buscando melhorias nas métricas associadas a disponibilidade.

Modelo Warm Standby no Nó

No tipo *warm standby* de redundância, um dos nós está operante, fornecendo o serviço, enquanto o outro está em *standby* aguardando para assumir o papel principal caso o primeiro

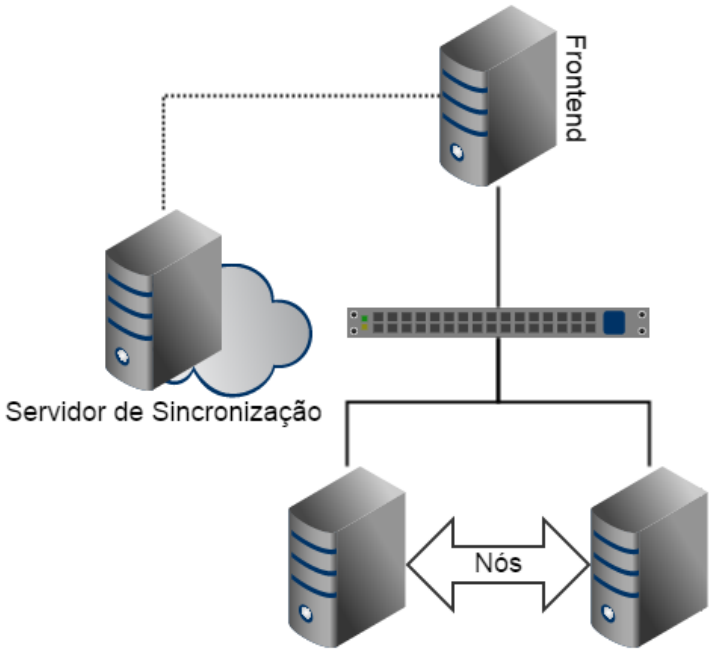


Figura 3.7: Visão de Alto nível da Arquitetura com nós redundantes

falhe, o que caracteriza uma prioridade entre componentes. Deste modo, para representação desta arquitetura uma nova CTMC é necessária e apresentada na Figura 3.8, baseada no modelo proposto por BEZERRA et al. (2014).

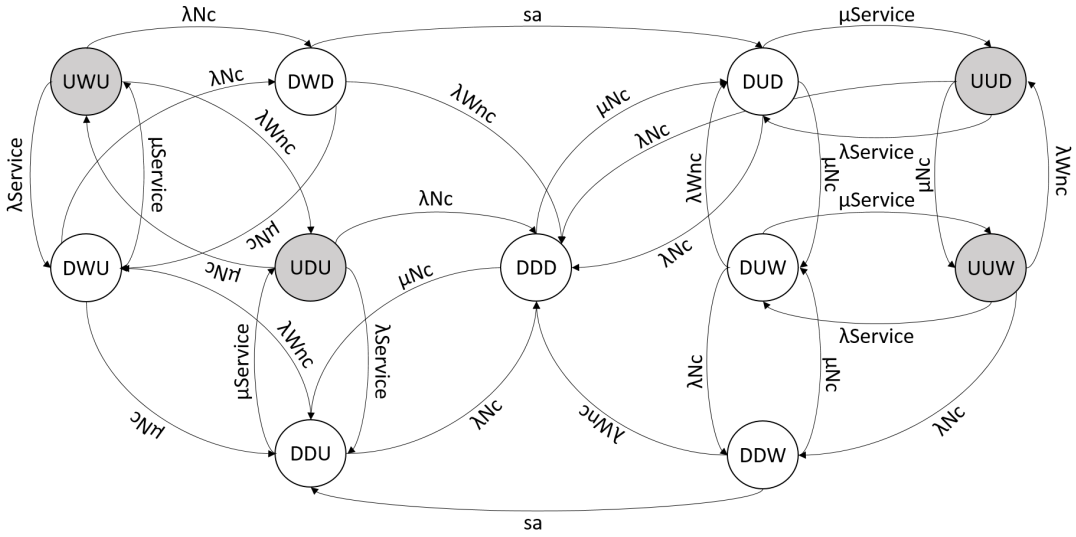


Figura 3.8: Arquitetura com nó em Warm Standby

As transições entre um estado e outro dentro desta CTMC são feitas através das ações de falhas e reparos descritos pela Tabela 3.7.

Diferentemente da CTMC do serviço, a notação adotada para definir o status dos componentes da CTMC dos nós se resume à **U** para disponível, **D** indisponível e **W** para aguardando; o primeiro caractere corresponde ao status do serviço, o segundo e o terceiro representam o nó em

Tabela 3.7: Taxas de Falha e Reparo dos Nós - CTMC

Taxa	Significado
$\lambda_{service}$	Falha do Serviço
$\mu_{service}$	Reparo do Serviço
λ_{Node}	Falha do Nó operante
μ_{Node}	Reparo do Nó operante
λ_{wNC}	Falha do Nó em standby
sa	Tempo de chaveamento do nó em standby

standby ou o nó onde o serviço encontra-se operante. Esta CTMC possui os seguintes estados: UWU, DWD, DWU, UDU, DDU, DDD, DUD, DUW, DDW, UUD e UUW. O sistema está disponível apenas se a CTMC estiver em um dos quatro estados a seguir: UWU, UDU, UUD e UUW, onde pelo menos um nó e o serviço estão operacionais.

A disponibilidade do sistema será igual a soma das probabilidades do sistema estar em um estado disponível. Os demais estados da CTMC são resumidos pela Tabela 3.8. Em nossa notação, o nó que estava em *standby* recebeu o nome de **Nó reserva**, enquanto que o nó que inicialmente fornecia o serviço foi intitulado de **Nó principal**

Tabela 3.8: Status da CTMC dos nós em warm standby

Estado	Significado	Status
UWU	Serviço UP, Nó reserva Aguardando, Nó principal UP	Disponivel
UUD	Serviço UP, Nó reserva UP, Nó principal DOWN	Disponível
UUW	Serviço UP, Nó reserva UP, Nó principal Aguardando	Disponível
UDU	Serviço UP, Nó reserva DOWN, Nó principal UP	Disponível
DUW	Serviço DOWN, Nó reserva UP, Nó principal Aguardando	Indisponível
DUD	Serviço DOWN, Nó reserva UP, Nó principal DOWN	Indisponível
DDD	Serviço DOWN, Nó reserva DOWN, Nó principal DOWN	Indisponível
DDW	Serviço DOWN, Nó reserva DOWN, Nó principal Aguardando	Indisponível
DDU	Serviço DOWN, Nó reserva DOWN, Nó principal UP	Indisponível
DWU	Serviço DOWN, Nó reserva Aguardando, Nó principal UP	Indisponível
DWD	Serviço DOWN, Nó reserva Aguardando, Nó principal DOWN	Indisponível

UWU representa o estado inicial da CTMC. A ocorrência de uma falha no serviço a uma taxa $\lambda_{Service}$ fará com que o sistema alcance o estado **DWU**, o serviço pode ser reparado a uma taxa $\mu_{Service}$, e retornar ao estado inicial. Ainda no estado **UWU** a ocorrência de falhas em ambos os nós é possível, seja a uma taxa λ_{NC} levando o sistema ao estado **DWD** ou a uma taxa λ_{wNC}, alcançando o estado **UDU**.

Se estivermos no estado **DWU** e um reparo não for possível, há chances de uma falha ocorrer a uma taxa λ_{WNc}, levando o sistema ao estado **DDU**, ou ainda no nó principal, chegando ao estado **DWD**.

Um outro possível cenário acontece a partir do estado **DWD**, onde se não houver reparo do nó principal nem o chaveamento do nó em espera, pode haver um colapso geral a uma taxa λ_{wNC}, o que nos leva ao estado **DDD**.

A partir do estado **UDU**, podemos voltar ao estado **UWU** a uma taxa μ_{Nc} ou o nó principal pode falhar levando ao estado **DDD** ou o serviço pode falhar, levando-nos ao estado **DDU**.

Já no estado **DDU** uma falha no nó principal leva ao estado **DDD**, e um reparo no serviço retorna o sistema a um estado disponível, **UDU**, caso o nó reserva seja reparado voltamos ao estado **DWU**.

O estado **DDD** representa o pior cenário possível, onde todos os componentes estão indisponíveis e devem ser reparados um a um.

Se no estado **DUD** um reparo no serviço nos leva ao estado **UUD**, enquanto que um reparo no nó principal nos leva ao estado **DUW**, ou, uma falha no nó reserva fará com que cheguemos ao estado **DDD**.

Já a partir do estado **DUW**, podemos alcançar o estado **UUW** através de um reparo no serviço, ou se uma falha acontecer no nó reserva o estado **DDW** é o que nos aguarda.

Partindo do estado **DDW**, é possível um reparo no nó reserva que leva ao estado **DUW**, o chaveamento do nó principal levando ao estado **DDU**, ou ainda uma falha no nó principal que nos leva ao estado **DDD**.

No estado **UUD**, uma falha no serviço nos leva ao estado **DUD** e o reparo no nó principal nos faz atingir o estado **UUW**, e há ainda a possibilidade de falha do nó reserva, o que mais uma vez leva ao estado **DDD**.

O último cenário é protagonizado pelo estado **UUW**, onde uma falha no nó principal nos leva ao estado **UUD**, ou uma falha no serviço nos faz alcançar o estado **DUW**, e há também a possibilidade de falha no nó reserva, esta que nos leva ao estado **DDW**.

A Expressão 3.3 representa a disponibilidade desta CTMC. Os termos α e β foram utilizados como fatores de simplificação para termos que se repetem na expressão, e equivalem respectivamente à $(\lambda_{nc}\lambda_{wnc})$ e $(\lambda_{wnc}+\mu_{nc})$, esta expressão foi extraída pela ferramenta Mathematica em conjunto do modelo desenvolvido na ferramenta Mercury.

$$A_{warm} = \frac{(2\lambda_{nc}(\alpha+(\beta)^2+sa(\lambda_{nc}+\beta))\mu_{service})}{((\lambda_{nc}(sa+\lambda_{wnc})(\alpha)+2(sa(\alpha)+\lambda_{wnc}(2\lambda_{nc}+\lambda_{wnc}))\mu_{nc}+2(sa+\lambda_{nc}+2\lambda_{wnc})(\mu_{nc})^2+(2\mu_{nc})^2)(\lambda_{nc}+\lambda_{service}+\mu_{service}))} \quad (3.3)$$

Outros tipos de redundância foram aplicadas aos nós, mas são apresentadas na Subseção 3.2.4, já que também se aplicam ao frontend, que tem como modelo exclusivo o proposto na Subseção 3.2.3, que assim como no caso do nó, trata-se de um modelo *warm standby*.

Redundância warm standby no Frontend

Arquiteturas que apresentam redundância no frontend terão um modo operacional diferente dos anteriormente propostos, sendo apresentado em detalhes no Capítulo 4 já que possui relação direta com os cenários que apresentam redundância nos nós. O modelo apresentado a seguir trata-se de uma exclusividade ao frontend em nossa arquitetura, por isso recebe este

destaque.

Na redundância *warm standby* para frontend, assim como nos nós, um dos componentes está operante, enquanto que o outro está em *standby*, aguardando para assumir o papel principal caso o primeiro falhe. A CTMC que representa o funcionamento do sistema é apresentada na Figura 3.9, baseada no modelo proposto por DANTAS et al. (2012a).

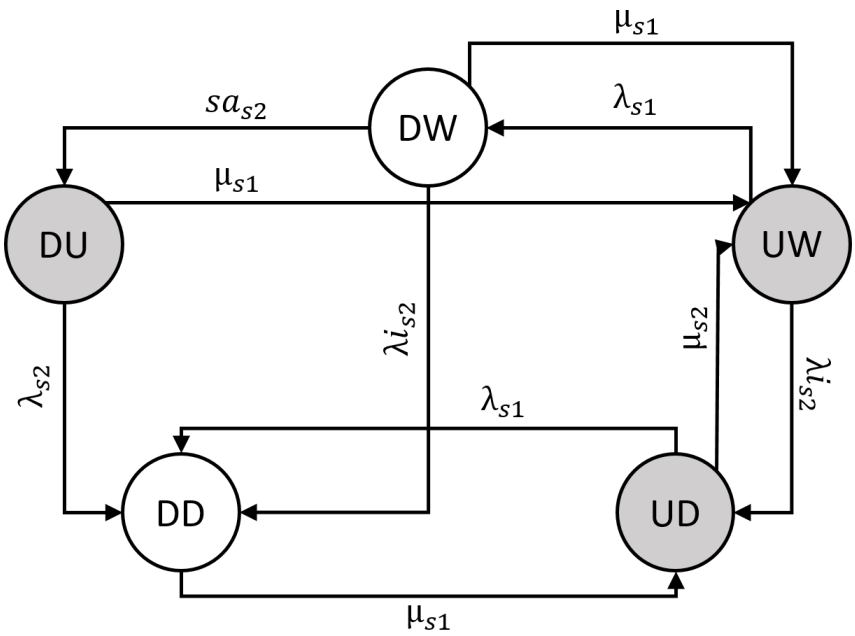


Figura 3.9: Modelo do frontend em Warm Standby (Baseado no proposto em DANTAS et al. (2012a))

A Expressão 3.4 apresenta como é feito o cálculo da disponibilidade para esta CTMC.

$$A_R = \frac{\mu(\lambda_i(\mu+sa)+\mu^2+sa(\lambda+\mu))}{\lambda_i(\lambda+\mu)(\mu+sa)+\mu^2(\lambda+\mu)+sa(\lambda^2+\lambda\mu+\mu^2)} \tag{3.4}$$

As taxas de falha e reparo dos frontends em *warm standby* são descritas pela Tabela 3.9 e, assim como na CTMC do nó em *warm standby*, a notação adotada para definir o *status* dos componentes desta CTMC são **U** para disponível, **D** indisponível e **W** para aguardando; o primeiro caractere corresponde ao status do frontend principal, enquanto que o segundo o que está inicialmente em *standby*, esta CTMC possui os seguintes estados: UW, DW, DU, UD e DD. O sistema estará disponível nos estados: UW, UU e DU, onde pelo menos um frontend está operacional. A Tabela 3.10 apresenta o resumo desta CTMC.

UW representa o estado inicial da CTMC. A ocorrência de falha no frontend principal a uma taxa λ_{s1}, o que fará com que o sistema alcance o estado **DW**, o frontend principal pode ser reparado a uma taxa μ_{s1}, e retornar ao estado inicial, ou o chaveamento do frontend em espera pode acontecer a uma taxa sa_{s2} levando o sistema ao estado **DU**, e a última possibilidade a partir daqui é a falha no frontend reserva a uma taxa λi_{s2}. Ainda no estado **DW**, é ainda possível a ocorrência de uma falha no frontend reserva a uma taxa λ_{s2} levando o sistema ao estado **DD**.

Tabela 3.9: Taxas de Falha e Reparo dos Frontends - CTMC

Taxa	Significado
μ_{s1}	Reparo do frontend principal
λ_{s1}	Falha do frontend principal
μ_{s2}	Reparo do frontend redundante
λ_{s2}	Falha do frontend redundante
Sa_{s2}	Tempo de chaveamento do frontend em espera

Tabela 3.10: Status da CTMC do frontend em warm standby

Estado	**Significado**	**Status**
UW	Frontend Principal UP, Frontend Reserva Aguardando	Disponível
DW	Frontend Principal DOWN, Frontend Reserva Aguardando	Indisponível
DU	Frontend Principal DOWN, Frontend Reserva UP	Disponível
UD	Frontend Principal UP, Frontend Reserva DOWN	Disponível
DD	Frontend Principal DOWN, Frontend Reserva DOWN	Indisponível

Ainda no estado **UW**, se uma falha ocorrer no frontend reserva, voltaremos ao estado **UD**.

Se estivermos no estado **DU**, um reparo no frontend reserva nos fará alcançar o estado **UW**, e uma falha no frontend reserva nos leva ao estado **DD**. No estado **DD** apenas o reparo do frontend principal é possível a uma taxa μ_{s1} nos levando ao estado **UD**.

Modelos genéricos de redundância (Frontend e Nó)

Este conjunto de modelos é aplicável tanto ao frontend quando ao nó com ajustes, apenas, em seus valores.

Modelo Hot Standby

Utilizando os RBDs propostos pela arquitetura básica, somos capazes de modelar a relação entre dois componentes em *Hot Standby*, através de diagramas de bloco de confiabilidade, pelo simples fato de não haver uma prioridade definida neste tipo de arquitetura, se um dos componentes em uma das máquinas falhar, a outra assumirá o papel principal, com um tempo mínimo de chaveamento e apresentado pela Figura 3.10.

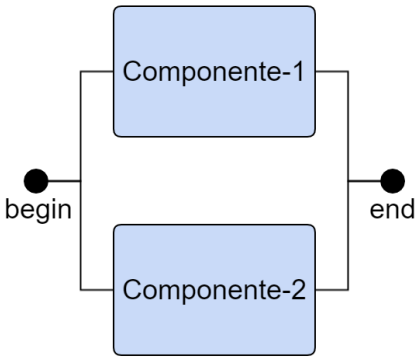


Figura 3.10: RBD de componentes em Hot Standby

Os submodelos para cada um dos blocos neste RBD são os mesmos apresentados na Subseção 3.2.1. A Expressão para cálculo da disponibilidade nesta arquitetura é dada por: $A = (A_{Componente-1} + A_{Componente-2})$.

Modelo Cold Standby

A Figura 3.11 apresenta a SPN para esta arquitetura, este modelo é baseado no apresentado em SOUSA et al. (2014).

Neste modelo, o componente principal está ativo e o reserva está em *standby*, aguardando a ocorrência de uma falha para assumir o protagonismo. O período que o componente reserva leva para assumir o papel principal em caso de ocorrência falha, é chamado de período de chaveamento, ou Mean Time To Activate (MTTA) KUO; ZUO (2003).

O modo operacional, ou expressão fechada que determina a disponibilidade dos componentes em *cold standby* é $MO = (ComponentePrincipal_{UP} = 0 \vee ComponenteReserva_{UP} = 0)$.

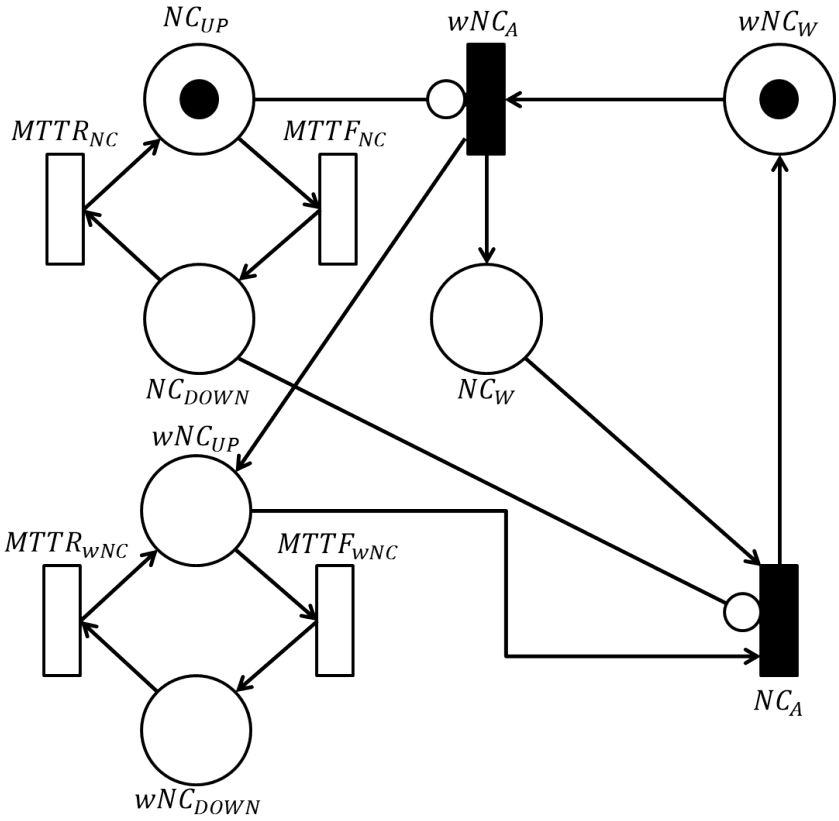


Figura 3.11: Modelo Cold Standby

Marcações nos lugares relacionados ao componente principal e aos componentes reservas determinam se um dado componente está disponível (**UP**) ou indisponível (**DOWN**). A transição temporizada $ComponenteReserva_{Ativo}$ representa o MTTA, enquanto que a marcação no lugar $ComponenteReserva_A$ indica que o nó reserva não está operacional.

Modelo para Disponibilidade Orientada à Capacidade

Com base nos valores obtidos pela avaliação de disponibilidade para cada modelo, estamos aptos para determinar a disponibilidade orientada à capacidade (COA), i.e., verificando a real quantidade de recursos disponíveis diante das rotinas de falha e reparo do sistema. Para este propósito, primeiramente precisamos saber o número de máquinas virtuais, suportado pelas máquinas físicas, descobrindo assim nossas limitações para o provimento do serviço.

Primeiramente escolhemos uma plataforma de computação em nuvem, na tentativa de identificar os recursos mínimos necessários para o provimento de máquinas virtuais neste tipo de ambiente. Novamente a plataforma Eucalyptus foi selecionada, nela somos capazes de dividir as máquinas virtuais em famílias, cada uma com sua própria característica HP (2015): Aplicações que demandam poucos recursos; com alta demanda de processamento; ou ainda, com grande utilização de RAM. Nossa escolha é pela primeira família, uma vez que nosso propósito é oferecer uma grande quantidade de máquinas virtuais, mas não com muitos usuários em cada, evitando a concorrência pelos recursos da VM.

A avaliação de disponibilidade orientada à capacidade, geralmente, se aplica a arquiteturas com redundância do tipo *hot standby*, onde os componentes podem estar dispostos de modo ativo-ativo, avaliando a degradação gradual do serviço e dos recursos gerais do computador. A SPN da Figura 3.12 apresenta o modelo de funcionamento para cada nó da arquitetura e seu número máximo de máquinas virtuais, o estado fundamental de falha é NODE_OFF, onde há a falha do nó, enquanto que o estado nVM_OFF representa o número de máquinas virtuais que já falharam, se todas as VM's entrarem em estado de falha, o serviço também se tornará indisponível. Em contrapartida nVM_ON apresenta o número de VM's disponíveis e NODE_ON representa a disponibilidade do nó.

A transição imediata **emptier** realiza o esvaziamento dos recursos de máquinas virtuais disponíveis e indisponíveis caso ocorra uma falha no nó onde estão em execução, o reparo dessas máquinas virtuais será feito automaticamente após reparo do nó, que levantará consigo o número máximo de máquinas virtuais por ele suportado.

A análise de COA irá relacionar a quantidade de máquinas virtuais suportadas e o número de nós disponíveis, com base no modelo apresentado na Figura 3.12 o serviço estará disponível se pelo menos uma marcação estiver presente nos estados nVM_ON e NODE_ON. A Expressão 3.5 representa como é realizado o cálculo da COA para este modelo.

$$COA = \frac{\sum_{i=1}^{NMVM}(P\{m(nVM_{on}) = i\} \times i)}{NMVM} \tag{3.5}$$

De acordo com a expressão, $m(nVM_{on}$ é a marcação do lugar nVM_{on}, que representa o conjunto de máquinas virtuais disponíveis no sistema em um estado s_i, e $NMVM$ representa a quantidade máxima de máquinas virtuais suportado por cada nó da arquitetura.

Este Capítulo apresentou os modelos utilizados no processo de avaliação de disponibi-

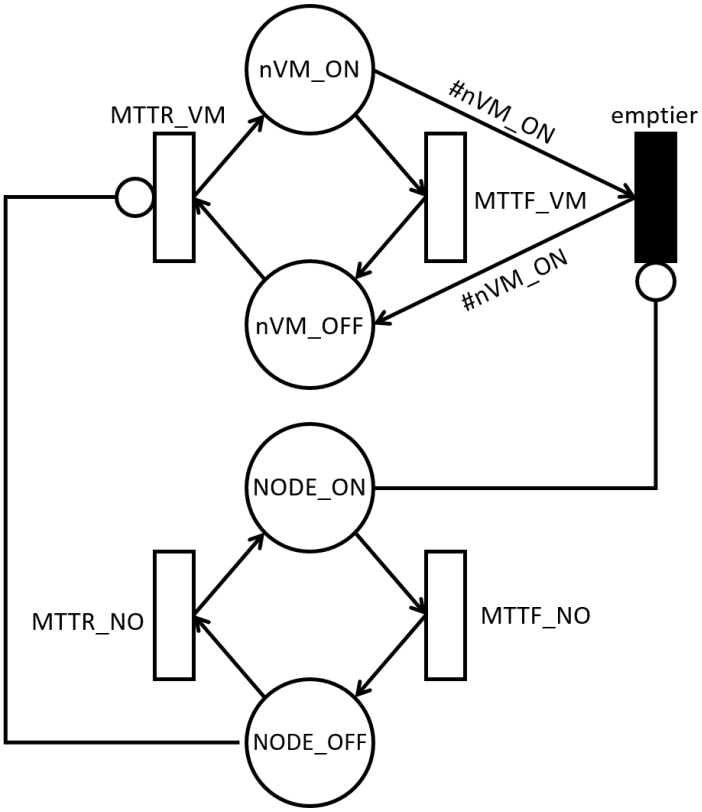


Figura 3.12: Modelo de Disponibilidade Orientada à Capacidade

lidade e a metodologia utilizada para que esta avaliação fosse realizada. O Capítulo a seguir apresenta os estudos de caso que fazem uso dos modelos aqui apresentados e demonstra o quão viáveis são.

4

Estudos de Caso

Este capítulo descreve estudos de caso que demonstram a utilidade dos modelos apresentados nesta dissertação. O primeiro estudo de caso consiste na avaliação de disponibilidade dos modelos que representam a arquitetura básica, contendo os componentes mínimos necessários ao fornecimento do serviço em uma infraestrutura baseada em Eucalyptus e, através da utilização de análise de sensibilidade que nos permitiu realizar o segundo estudo de caso, onde detectamos os componentes de maior impacto em nossa métrica de interesse e utilizamos de redundância como um método para alta disponibilidade;

No terceiro estudo de caso um modelo para avaliação de disponibilidade orientada à capacidade em um ambiente com nós em redundância ativo-ativo também é analisado, mostrando o real impacto das falhas nos recursos utilizados pelo serviço; uma série de outros modelos e variações da arquitetura básica também é proposta, além da validação do componente responsável pela sincronização de dados, através de experimentos que consistiram no monitoramento e injeção de falhas na aplicação.

No quarto e último estudo de caso, o custo de implantação para cada arquitetura proposta é levantado, este valor engloba além do custo de aquisição dos equipamentos necessários ao fornecimento do serviço, o custo energético esperado para cada infraestrutura no intervalo de um ano, uma relação entre este custo e o *downtime* anual das arquiteturas também é apresentada, apontando assim, as arquiteturas que atendam melhor as necessidades do gestor da infraestrutura.

Estudo de Caso I - Avaliação de Disponibilidade da Arquitetura Básica

Neste estudo de caso são apresentadas as avaliações dos modelos de disponibilidade propostos no Capítulo 3, e que representam os modos operacionais do serviço de sincronização de dados hospedado na arquitetura básica, com um frontend, responsável pela gerência e acesso a infraestrutura e um nó, responsável pelo fornecimento das máquinas virtuais, onde o serviço estará em execução. É importante ressaltar que os modelos propostos foram apresentados de forma hierárquica, onde na arquitetura básica uma CTMC representa o serviço e um RBD

descreve a infraestrutura necessária para que o serviço seja fornecido.

Para a avaliação dos modelos baseados em estado propostos na Subseção 3.2.1, e que representam o funcionamento da arquitetura básica, alguns valores de entrada tornaram-se necessários. Estes valores são apresentados na Tabela 4.1, alguns deles foram extraídos de trabalhos realizados por DANTAS et al. (2012b); COSTA et al. (2015); BEZERRA et al. (2014); CAMPOS et al. (2015); SOUSA et al. (2014) outros calculados a partir dos modelos propostos para a arquitetura básica, este é o caso do SyncML, que teve seu MTTF estimado com base em outras aplicações presentes em nossas arquiteturas como o Apache, e teve este valor validado através da realização de experimentos. A avaliação de disponibilidade teve como período pré-determinado de um ano ou 8760 horas.

Tabela 4.1: Parâmetros de Entrada para os modelos

Componentes	**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
Hypervisor	2990 h	1 h
NC	788,4 h	1 h
SyncML	788,4 h	1 h
Apache	788,4 h	1 h
Database	1440 h	3 h
VM	2880 h	69 s

Com base nos parâmetros de entrada, pudemos extrair como saída valores relacionados à disponibilidade do RBD que representa frontend e nó (Figura 3.3), e a CTMC do serviço que compõe a arquitetura básica (Figura 3.6), a Tabela 4.2 apresenta um resumo dos resultados obtidos pela avaliação.

Tabela 4.2: Resultados da Avaliação de Disponibilidade para os modelos da arquitetura básica

Modelos / Métrica	**RBD do Frontend**	**RBD do Nó**	**CTMC do Serviço**
Disponibilidade (%)	99,46	99,81	99,53
Disponibilidade (Número de Noves)	2,27	2,72	2,33
Uptime anual (h)	8713,22	8743,53	8719,63
Downtime anual (h)	46,78	16,47	40,37

Para o RBD que representa o subsistema do frontend (Figura 3.4) um valor que corresponde a 99,46% de disponibilidade no decorrer de um ano é apresentado, e um *downtime* anual de 46,78 horas. Já para o subsistema representado pelo RBD do nó (Figura 3.5) este valor alcança

99,81%, garantindo um *downtime* anual de dezesseis horas, quase um terço do apresentado pelo RBD do frontend e, menos de um dia de indisponibilidade; isto acontece em virtude do número de aplicações em execução em cada componente, enquanto que o RBD que representa o subsistema do nó contém apenas o hypervisor e o Controlador do Nó como aplicações, já que o serviço foi extraído para uma CTMC à parte, o RBD do subsistema do frontend abriga os controladores de Nuvem, Armazenamento, Cluster e Walrus, garantindo assim uma maior probabilidade de falha.

A avaliação de disponibilidade da CTMC do serviço resultou em uma disponibilidade de 99,53%, implicando em um *downtime* anual de 40 horas, graças ao tempo necessário para realização de reparo neste componente, que é 2,5 maior que o reparo do nó com o serviço à parte.

De posse dos resultados obtidos, é possível encontrar o valor de 98,82% de disponibilidade para a arquitetura básica (A1), ou 103,14 horas de *downtime* anual, o que corresponde a um período de quatro dias, onde os usuários do serviço não poderão realizar a sincronização de dados.

Análise de sensibilidade da Arquitetura Básica

Os valores anteriormente utilizados como entrada para a avaliação de disponibilidade dos modelos propostos agora são transformados nos parâmetros de entrada para a análise de sensibilidade e relacionados diretamente ao valor de disponibilidade obtido. A Tabela 4.3 apresenta os parâmetros de entrada para a nova análise de sensibilidade.

Tabela 4.3: Parâmetros de entrada para análise de sensibilidade das arquiteturas propostas

Parâmetros	Valores
$MTTF_{NC}$, $MTTF_{walrus}$, $MTTF_{SC}$, $MTTF_{CC}$, $MTTF_{CLC}$ (h)	788,4
$MTTF_{hypervisor}$ (h)	2990
λ_{syncml}, λ_{apache} (h^{-1})	$1/788.4$
λ_{db} (h^{-1})	$1/1440$
μ_{db} (h^{-1})	$1/3$
reboot (h^{-1})	$1/0.019166$
λ_{VM} (h^{-1})	$1/2880$
$MTTR_{NC}$, $MTTR_{walrus}$, $MTTR_{SC}$, $MTTR_{CC}$, μ_{syncml}, μ_{apache} (h)	1
$MTTR_{SO}$ (h)	0,25
$MTTF_{SO}$ (h)	2893
$MTTR_{HW}$ (h)	1,66
$MTTF_{HW}$ (h)	8760

É importante ressaltar que a técnica utilizada para a análise de sensibilidade foi a de diferença percentual, que consiste na variação de um parâmetro por vez, enquanto os demais são fixados, os resultados da análise de sensibilidade podem ser vistos na Tabela 4.4 que aponta um *ranking* ordenado pelo impacto causado por cada componente na disponibilidade do sistema.

Alguns dos componentes no topo do *ranking* de sensibilidade estão relacionados de

Tabela 4.4: Ranking de sensibilidade para a Arquitetura Básica

Componente	Posição	Valor
λ_{DB}	1º	$1,05 \times 10^{-}2$
μ_{DB}	2º	$8,47 \times 10^{-3}$
$MTTF_{NC}$, $MTTF_{walrus}$, $MTTF_{CC}$, $MTTF_{SC}$, $MTTF_{CLC}$	3º	$1,69 \times 10^{-3}$
λ_{apache}, λ_{syncml}	7º	$1,50 \times 10^{-3}$
μ_{apache}, μ_{syncml}	9º	$1,26 \times 10^{-3}$
$MTTR_{NC}$, $MTTR_{walrus}$, $MTTR_{CLC}$, $MTTR_{SC}$, $MTTR_{CC}$	11º	$1,13 \times 10^{-3}$
$MTTF_{hypervisor}$	16º	$4,46 \times 10^{-4}$
$MTTR_{hypervisor}$	17º	$2,97 \times 10^{-4}$
$MTTF_{HW_frontend}$, $MTTF_{HW_no}$	18º	$2,42 \times 10^{-4}$
$MTTF_{SO_frontend}$, $MTTF_{SO_no}$	20º	$1,15 \times 10^{-4}$
$MTTR_{HW_frontend}$, $MTTR_{HW_no}$	22º	$1,01 \times 10^{-4}$
$MTTR_{SO_frontend}$, $MTTR_{SO_no}$	24º	$7,68 \times 10^{-5}$
λ_{vm}	25º	$4,05 \times 10^{-5}$
reboot	26º	$6,66 \times 10^{-6}$

maneira direta ou indireta com o serviço e nó, onde o mesmo está em execução, a variação dos valores e taxas destes componentes podem afetar significativamente a disponibilidade do serviço, é o caso da taxa de falha do banco de dados λ_{DB} e da taxa de reparo μ_{DB} que são componentes do serviço.

Os resultados das variações destes componentes podem ser vistos nas Figuras 4.1 e 4.2, onde a linha pontilhada representa a disponibilidade do sistema com os valores iniciais e a linha contínua o valor que a disponibilidade poderia assumir, caso os tempos e taxas fossem modificadas.

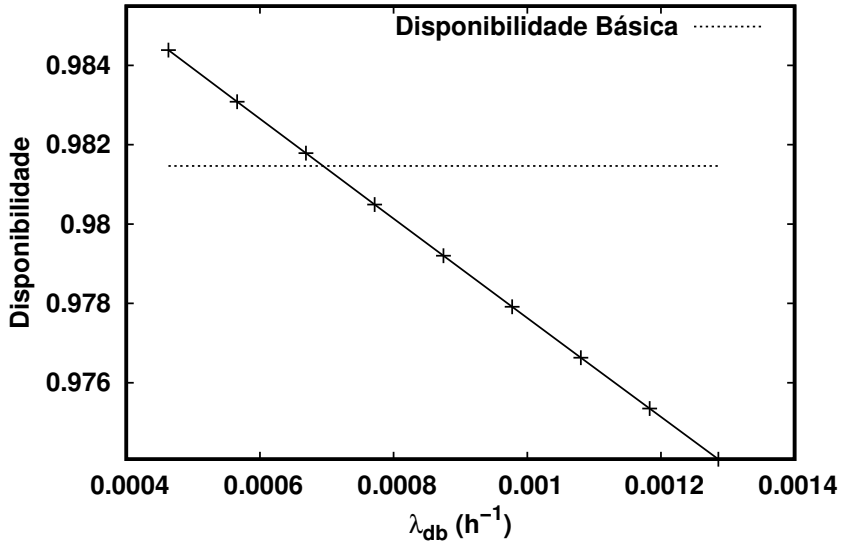


Figura 4.1: Variação da taxa de falha λ_{db}

Se de alguma forma conseguirmos reduzir a taxa de falha λ_{DB} poderemos obter uma

melhora na disponibilidade do sistema.

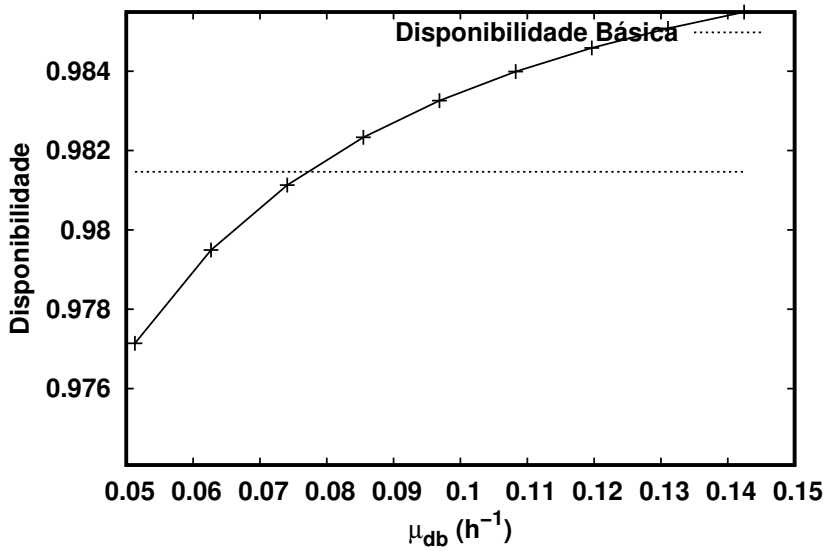


Figura 4.2: Variação da taxa de reparo μ_{DB}

Já no segundo caso, se aumentarmos a taxa de reparo μ_{DB} nós teremos um acréscimo na disponibilidade do sistema, o que vem a ser uma questão de lógica: Quanto mais alto o tempo de reparo, mais tempo de indisponibilidade teremos, já em relação ao tempo para falha, quanto maior for seu valor, maior será o período de disponibilidade. É importante salientar que os valores das taxas correspondem ao inverso do valor do tempo, ou seja, $\frac{1}{tempo}$.

Todavia, esta lógica sobre acréscimos e decréscimos é apenas aplicável aos componentes que realmente importam e que são encontrados graças a análise de sensibilidade, em outro cenário, temos a taxa *reboot*,o componente em última colocação no *ranking*, como pode ser visto na Figura 4.3.

A variação para o tempo de reinicialização da Máquina Virtual pouco afetará na disponibilidade do nosso serviço de sincronização de dados. Resolvemos aplicar técnicas de redundância visando uma melhoria da disponibilidade total do sistema com base no impacto dos parâmetros, criando e sugerindo novos cenários e novas arquiteturas.

Primeiramente, aplicamos redundância ao subsistema serviço, onde encontram-se as aplicações e a máquina virtual, optamos pela redundância do tipo ativo-ativo, onde variamos de dois à dezesseis blocos do componente serviço, que operam paralelamente e oferecem aos usuários o mesmo simultaneamente, em caso de falha de uma delas a outra continuará a atender os usuários, utilizamos da linguagem de *scripts* da ferramenta Mercury para avaliar hierarquicamente este cenário. Os resultados são apresentados na Tabela 4.5.

Com base nos resultados apresentados pela Tabela 4.5 que o número de redundâncias do componente proporciona um aumento na disponibilidade até um certo ponto, que é prorrogado com estagnação deste valor, ou pouca variação, para uma redundância com dois blocos serviços a disponibilidade salta de 98,82% para 99,26%, outro salto ocorre quando temos quatro com-

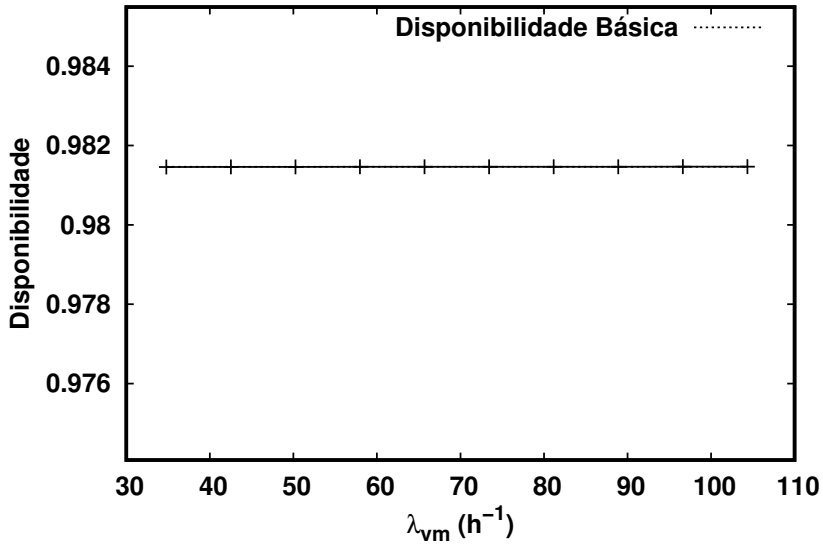


Figura 4.3: Variação da taxa *reboot*

Tabela 4.5: Disponibilidade para o serviço redundante

Número de Componentes Redundantes	Disponibilidade
2	0,992672
4	0.992801
8	0.992801
16	0.992801

ponentes em redundância, saltando para 99,28%, porém, a partir deste ponto não há melhora significativa, o que sugere a existência de correlação entre os componentes no topo do *ranking*, isto é avaliado e posto em prática na próxima seção, onde a replicação no nó como um todo é avaliada.

Estudo de Caso II - Sugestão de novas arquiteturas variantes da arquitetura básica

Ao todo, três arquiteturas com redundância nos nós são apresentadas na Subseção 3.2.2, uma visão de alto nível mostra como a arquitetura com dois nós e um frontend se comporta (Figura 3.7). As redundâncias do tipo *hot standby* (Figura 3.10), *warm standby* (Figura 3.8) e *cold standby* (Figura 3.11), utilizam como base os valores de MTTF e MTTR apresentados na Tabela 4.1.

A aplicação das técnicas de redundância no nó da arquitetura básica se justificam pelos valores concebidos através da análise de sensibilidade realizada e apresentados na Tabela 4.4, indicando a necessidade da melhoria nos valores das métricas dos componentes pertencentes ao serviço.

Nesta fase temos inicialmente a arquitetura com nó em *hot standby* (A2), que apresenta um valor de disponibilidade de aproximados 99,46%, um grande incremento caso comparada a arquitetura básica com seus 98,82%.

A arquitetura A2 também apresenta um valor de disponibilidade maior que as outras duas arquiteturas com redundâncias aplicadas ao nó: *warm standby* (A3) que apresenta 99,44% de disponibilidade e *cold standby* (A4) com 99,40%. Estas "pequenas"variações entre A2, A3 e A4 decorrem do tempo necessário para o chaveamento MTTA ou inicialização do sistema em uma máquina que está em estado *standby*, este que para a arquitetura em *warm* teve o valor aproximado de 2 minutos e na arquitetura em *cold standby* de 10 minutos SOUSA et al. (2014).

Para os próximos avanços uma nova análise de sensibilidade precisou ser aplicada, já que almejamos, pelo menos 99,9% de disponibilidade, novamente identificamos quais os componentes de maior impacto para esta métrica. Como os componentes de cada arquitetura com nó redundante são distintos, ou seja, o serviço pode ser apresentado como um único componente ao lado do nó ou como diversos componentes distintos, optamos por realizar esta atividade apenas na arquitetura que apresentou o maior valor de disponibilidade: A2.

Os valores de entrada para análise de sensibilidade da arquitetura A2 são os mesmos que foram utilizados na arquitetura A1. Enquanto que os resultados da análise de sensibilidade da segunda arquitetura são generalizados e apresentados na Tabela 4.6 à seguir.

Tabela 4.6: Ranking de sensibilidade para arquitetura com nó hot standby

Componente	Posição	SS(A)
$MTTF_{frontend}$	1º	$7,07 \times 10^{-3}$
$MTTR_{frontend}$	2º	$5,32 \times 10^{-3}$
$MTTF_{no1,no2}$	3º	$5,17 \times 10^{-6}$
$MTTR_{no1,no2}$	4º	$3,88 \times 10^{-6}$

É notória a importância dos componentes pertencentes ao frontend, visto que os seus valores de MTTF e MTTR estão no topo do novo *ranking*. E foi através desta análise que pudemos concluir que a aplicação de técnicas de redundância neste componente poderiam novamente alavancar os valores de disponibilidade das arquiteturas.

Um total de 9 arquiteturas são propostas de acordo com os valores de análise de sensibilidade aplicados ao modelo com redundância *hot standby* nos nós. Os três tipos de redundância anteriormente apresentados são aplicados ao frontend e a nó, e o modo operacional passa a ser descrito pela expressão: $MO_{sistema} = (frontend1 \wedge frontend2) \vee (no1 \wedge no2)$.

A linguagem de *script* da ferramenta Mercury foi novamente utilizada para dispor os modelos apresentados no Capítulo anterior de maneira hierárquica. As subseções à seguir apresentam os resultados de disponibilidade para nove combinações de arquiteturas distintas.

A Figura 4.4 apresenta uma visão de alto nível do que propomos, agora quatro máquinas compõe nossa arquitetura: dois frontends e dois nós.

Os resultados para o modelo hierárquico composto por frontend e nó em *hot standby*, ou

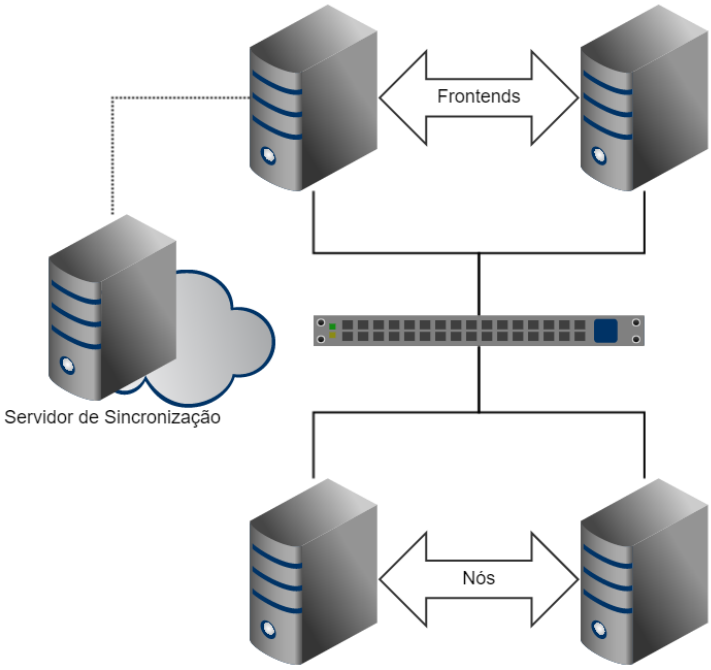


Figura 4.4: Visão de Alto nível da Arquitetura com frontend e nós redundantes

arquitetura (A5) demonstram uma disponibilidade de 99,996% e 0,28 horas ou 17 minutos de *downtime* no decorrer de um ano, sendo esta a primeira das arquiteturas a ultrapassar a marca de 3 noves de disponibilidade, atingindo 4,49 noves.

Levando-se apenas disponibilidade em consideração a arquitetura A5 merece o grande destaque, ela se sobressai sobre as demais ao apresentar o maior número de noves dentre todas elas. Já o destaque negativo em termos de disponibilidade unicamente, vai para a arquitetura A13, com nó e frontend em *cold standby*, em virtude principalmente do período de chaveamento em ambos os subsistemas, a disponibilidade ficou em 99,86% e o *downtime* 43,32 vezes maior que o da arquitetura A5.

A Tabela 4.7 descreve cada arquitetura proposta e apresentada nesta seção, o número demáquinas utilizáveis e o tipo de redundância aplicada em cada componente, além do *id* que a identifica.

A Tabela 4.8 apresenta os resultados de disponibilidade e *downtime* anual para todas as arquiteturas propostas, e resume tudo aquilo que foi apresentado até o momento

A próxima seção avalia um cenário onde dois nós estão ativos ao mesmo tempo e recebendo carga de trabalho, funcionando paralelamente de forma similar a apresentada pela Figura 3.10, o objetivo desta nova análise é avaliação do real impacto das falhas no fornecimento do serviço de sincronização de dados.

Tabela 4.7: Lista de Arquiteturas

id	Resumo da Arquitetura	Nº de Máquinas
A1	Arquitetura Básica	2
A2	Nó em Hot Standby	3
A3	Nó em Warm Standby	3
A4	Nó em Cold Standby	3
A5	Frontend e Nó em Hot Standby	4
A6	Frontend em Hot Standby e Nó em Warm Standby	4
A7	Frontend em Hot Standby e Nó em Cold Standby	4
A8	Frontend em Warm Standby e Nó em Hot Standby	4
A9	Frontend e Nó em Warm Standby	4
A10	Frontend em Warm Standby e Nó em Cold Stanby	4
A11	Frontend em Cold Standby e Nó em Hot Standby	4
A12	Frontend em Cold Standby e Nó em Warm Standby	4
A13	Frontend e Nó em Cold Standby	4

Tabela 4.8: Comparação entre os resultados de disponibilidade para cada arquitetura redundante

Métrica / Arquitetura	Disponibilidade (%)	Nº de 9	Uptime (hrs/ano)	Downtime (hrs/ano)
A1	98,82	1,92	8656,86	103,14
A2	99,46	2,26	8713,26	46,74
A3	99,44	2,25	8711,37	48,63
A4	99,40	2,22	8707,9	52,10
A5	99,996	4,49	8759,72	0,28
A6	99,975	3,60	8757,82	2,18
A7	99,935	3,18	8754,33	5,67
A8	99,991	4,08	8759,28	0,72
A9	99,970	3,52	8757,38	2,62
A10	99,930	3,15	8753,89	6,11
A11	99,923	3,11	8753,26	6,74
A12	99,901	3,00	8751,27	8,63
A13	99,861	2,86	8747,87	12,13

Estudo de Caso III - Disponibilidade Orientada à Capacidade

Primeiramente é preciso definir o que vem a ser nossa capacidade. A priori, este estudo avalia como capacidade o número de máquinas virtuais suportado por cada computador do ambiente configurado pela plataforma Eucalyptus.

Os valores de entrada para a avaliação de COA são os mesmo que foram apresentados na Tabela 4.1. De posse dos dados de entrada e das configurações para cada família de máquina virtual que pode ser criada na plataforma Eucalyptus, podemos apresentar a Tabela 4.9, retirada de HP (2015).

As máquinas virtuais do tipo M1 e M3 são de propósito geral e aceitáveis por aplicações que demandem pouca quantidade de recursos, tendo como valor máximo de armazenamento 30GB e quatro núcleos com 4GB de RAM. Já as do tipo C1, CC1, CC2 destinam-se a aplicações

Tabela 4.9: Tipos e Configurações de VM criáveis com Eucalyptus

Tipos de VM	CPU's Utilizáveis	Disco (GB)	RAM (MB)
m1.small	1	5	256
m1.medium	1	10	512
m1.large	2	10	512
m1.xlarge	2	10	1024
m2.xlarge	4	15	2048
m3.2xlarge	4	30	4096
c1.medium	2	10	512
c1.xlarge	2	10	2048
cc1.4xlarge	8	60	3072
cc2.8xlarge	16	120	6144
m2.xlarge	2	10	2048
m2.2xlarge	2	30	4096
m2.4xlarge	8	60	4096
cr1.8xlarge	16	240	16384
t1.micro	1	5	256

com alto consumo de processamento. Instâncias do tipo M2 e CR1 são voltadas a aplicações que necessitem de mais recursos de RAM, enquanto que as do tipo T1 são as mais simples e voltadas a aplicações como *proxy* e *gateway* de infraestruturas.

De posse de cada tipo de VM criável, pudemos calcular o número de máquinas virtuais criáveis por cada nó, tendo como computador base um Dell PowerEdge T320 com Intel Xeon E5-2420, com seis núcleos físicos e doze *threads*, 1TB de disco, e 24GB de RAM. É importante ressaltar que o virtualizador nativo do Eucalyptus, o KVM, proporciona a criação de uma máquina virtual por *thread* disponível. Outro estudo viável é a determinação do número de usuários suportado por cada tipo de máquina virtual, para tanto; tomamos como base a conta básica de um já conhecido serviço de sincronização: O DropBox, onde cada usuário tem 2GB de armazenamento disponível. Cada VM tem como reservados 5GB para o sistema operacional e aplicações relacionadas ao sincronizador. Os valores são apresentados na Tabela 4.10.

O número de usuários em VM's do tipo m1.small e t1.micro é zero, dado o fato que estabelecemos um valor mínimo de 5GB de disco para sistema operacional e aplicações relacionadas ao provimento do serviço de sincronização de dados. Para as demais a quantidade de usuários varia de 15 a 88,12 usuários.

Doze máquinas virtuais do tipo m1.medium podem ser criadas por nó, em uma arquitetura com dois nós redundantes em *hot standby*, por exemplo, teremos 24 máquinas disponíveis e qualquer valor abaixo disso é uma degradação decorrente da ocorrência de falha ou de uma rotina de reparo. Os resultados iniciais apontam uma disponibilidade orientada à capacidade de 99,78%, isto significa que aproximadamente 0.22% das máquinas virtuais, ou recursos, são perdidos por conta de falhas, para um cenário com 24 máquinas virtuais, cerca de 23,95 estarão utilizáveis, de acordo com o modelo apresentado Figura 3.12. Quanto ao número de usuários

Tabela 4.10: Número de VMs e Usuários por Nó

Tipos de VM	Nº de VM's Criáveis por Nó	Nº de Usuários por VM	Nº de Usuários por Nó
m1.small	12	0	0
m1.medium	12	2,5	30
m1.large	6	2,5	15
m1.xlarge	6	2,5	15
m2.xlarge	3	5	15
m3.2xlarge	3	12,5	37,5
c1.medium	6	2,5	15
c1.xlarge	6	2,5	15
cc1.4xlarge	1,5	27,5	41,25
cc2.8xlarge	0,75	57,5	43,12
m2.xlarge	6	2,5	15
m2.2xlarge	6	12,5	75
m2.4xlarge	1,5	27,5	41,25
cr1.8xlarge	0,75	117,5	88,12
t1.micro	12	0	0

suportados, as VM's do tipo m1.medium perderão até 2,5 usuários por nó, com a perda de até uma máquina virtual. A Tabela 4.11 apresenta um resumo da relação entre COA e número de máquinas virtuais e suas famílias por nó da arquitetura.

Tabela 4.11: Eucalyptus - COA (Disponibilidade $\times$ VM)

Tipo de VM	Número de VM por nó	Número de VMs perdidas	COA
m1.small	12	0,05	23,95
m1.medium	12	0,05	23,95
m1.large	6	0,03	11,97
m1.xlarge	6	0,03	11,97
m2.xlarge	3	0,02	5,98
m3.2xlarge	3	0,02	5,98
c1.medium	6	0,03	11,97
c1.xlarge	6	0,03	11,97
cc1.4xlarge	1,5	0,01	1,99
cc2.8xlarge	0,75	-	-
m2.xlarge	6	0,03	11,97
m2.2xlarge	6	0,03	11,97
m2.4xlarge	1,5	0,01	1,99
cr1.8xlarge	0,75	-	-
t1.micro	12	0,05	23,95

Os nós podem suportar mais máquinas virtuais dos tipos m1.small, m1.medium e t1.micro, as VMs do tipo m1.small e m1.medium suportam tantos usuários quanto as do tipo m1.large, m1.xlarge e m2.xlarge, sendo ótimas escolhas ao nosso estudo de caso. Todavia, elas não suportam um número de usuários tão grande quanto os suportados pelas VMs dos tipos m2.2xlarge, que suportam um número de usuários cinco vezes maior, e apresentam os mesmos valores de COA que as VMs dos tipos m2.xlarge, c1.medium, c1.large, m1.large, and m1.xlarge. Este estudo oferece a melhor escolha para os provedores de serviço de sincronização

Tabela 4.12: Análise de Sensibilidade para modelo de COA

Parâmetro	S(COA)
$MTTF_{NODE}$	$2,83 \times 10^{-3}$
$MTTR_{NODE}$	$1,66 \times 10^{-3}$
$MTTR_{VM}$	$2,32 \times 10^{-4}$
$MTTF_{VM}$	$8,60 \times 10^{-6}$

que possuam como gargalo do sistema o dispositivo de armazenamento, mas é preciso analisar as vantagens e desvantagens antes de tomar uma decisão final.

O último passo de nossa análise de disponibilidade é encontrar e determinar qual/quais componentes possuem um maior impacto nesta análise: Máquinas Virtuais ou Nó. A diferença percentual foi novamente escolhida como técnica de análise de sensibilidade, seguindo as mesmas regras e valores apresentadas na Tabela 4.3, os resultados obtidos são apresentados na Tabela 4.12, e representam a análise de sensibilidade aplicada ao modelo da Figura 3.12.

A Figura 4.5 apresenta o relacionamento entre tempo médio para falha do nó e a disponibilidade orientada a capacidade, quanto mais tempo o nó leva pra falhar, maior será a quantidade de recursos disponível.

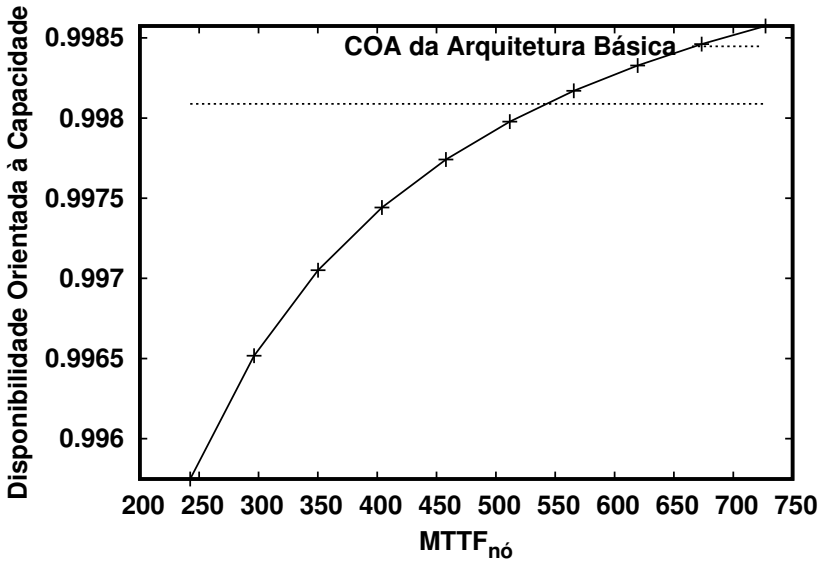


Figura 4.5: Variação do Nó

O índice de sensibilidade do MTTF do nó é seguido de perto pelo seu MTTR, o que significa que quanto maior o número de nós, maior a quantidade de recursos disponível, fazendo com que a aplicação da COA apenas faça sentido nesse tipo redundância. Já os tempos médios de falha e reparo das máquinas virtuais possuem um valor de impacto inferior, mas não menos importante que os outros componentes, e devem ser considerados.

Na Seção 4.4 é realizada uma análise de custos para implantação de cada uma das arquiteturas anteriormente propostas, além de uma relação com o *downtime* anual por elas apresentado.

Visando a apresentação do melhor custo × benefício para os gestores de infraestruturas de computação em nuvem e fornecedores de serviços de sincronização.

Estudo de Caso IV - Análise de Custo de Implantação × Downtime Anual

Este estudo de caso apresenta os custos de implantação para cada uma das treze arquiteturas propostas e, analisa a relação entre este custo e os valores de *downtime* anual para cada uma delas. O principal objetivo é detectar qual a melhor arquitetura para um administrador implantar em sua empresa, para isso ele precisa estabelecer o que é mais importante na situação atual da empresa: recursos monetários ou alta disponibilidade do serviço, ou ainda, optar por um meio termo entre ambas as métricas.

Para a realização da análise de custos de implantação uma série de dados precisou ser levantada, este levantamento foi feito em três etapas. A primeira etapa consistiu na delimitação dos componentes necessários ao provimento do serviço, boa parte deste processo foi realizado durante a definição das arquiteturas no primeiro estudo de caso. Deste modo, podemos definir os componentes de cada infraestrutura como: Computadores (número variável por arquitetura), *switch*, *racks*, monitores, teclado, mouse, computador pessoal e ar condicionado (número fixo).

Na segunda etapa, o custo de aquisição para cada componente precisou ser analisado individualmente, sites de compras foram consultados, alguns deles nacionais e outros internacionais, o menor valor dos componentes no dia 13 de Abril de 2016 foi o escolhido.

O preço do servidor Dell PowerEdge T320 teve seu menor valor encontrado (US$1.209,82) na loja norte americana **ant online**. Já a loja brasileira Central Ar, o Ar condicionado Consul de 12000 BTUs se encontrava por R$1.729,00, e que pela cotação do dólar naquele mesmo dia, este valor pode ser convertido para aproximados US$490,83. Já o *switch* Dell 2808 se encontrava por US$119 na loja oficial da fabricante. Uma torre de computador pessoal na loja brasileira Kabum por 313 dólares, um teclado de baixo custo por US$5,71 na mesma loja, um mouse também de baixo custo no valor de US$2,55 e um monitor US$191,95 também no Kabum e, por fim, um rack de 24U que suporta todas as arquiteturas e foi avaliado em US$416,73 na loja Tecno Ferramentas. Todo o software em execução é gratuito e *open source*.

A terceira etapa se caracteriza pela a análise da especificação técnica de cada componente em busca dos valores de potência energética, objetivando o cálculo do custo energético unitário para o período de um ano, onde estes componentes estariam funcionais 24 horas por dia os 7 dias da semana. O custo do killowatt hora na cidade de Recife, em março de 2016, foi de 58 centavos de reais, que pela cotação do dia 13 de abril, pode ser convertido a 16 centavos de dólar, de posse destes dados somos capazes de calcular a quantidade de kwh consumidos por um dispositivo no decorrer de um ano, através da Expressão 4.1, e multiplicar o seu resultado pelo valor do kwh. Nesta expressão **W** equivale a potência do equipamento, **ND** ao número total de dias em

que o mesmo foi utilizado e, por fim, a métrica **NH**, que representa o total de horas de uso do equipamento.

$$kwh = \frac{W \times NH \times ND}{1000} \tag{4.1}$$

A Tabela 4.13 apresenta a relação entre os componentes e seus custos de aquisição, valor de potência e custo energético anual.

A Tabela 4.14 apresenta os custos gerais de aquisição para cada arquitetura proposta, onde a coluna que corresponde ao valor total tratando da soma dos custos de aquisição e das despesas com eletricidade com cada uma delas.

Algumas despesas são fixas para cada arquitetura, estas despesas correspondem ao *switch*, ar condicionado, monitor, teclado, mouse e a torre do computador pessoal necessárias para montagem e acesso à infraestrutura, um custo de US$1.540,04. Outra despesa fixa é do consumo energético destes componentes, esta que é de US$1.761,67.

A arquitetura com o maior custo é a A5, que para sua implementação e funcionamento no decorrer de um ano apresenta um custo de US$8.701,63 dólares é necessário, um valor US$1.700,00 maior que o apresentado pela arquitetura de menor custo: A1. Na arquitetura A5 todas as cinco máquinas estarão ligadas e operacionais 24 horas por dia os 7 dias da semana. É importante informar também os valores em watts para as arquiteturas com redundância de tipo *warm* e *cold standby*, que foram de 48,7 e 3,75 watts respectivamente. Estes valores foram encontrados através de um experimento, que consistiu na medição dos valores em watts por máquinas em modo *standby* e desligadas através do dispositivo WattsUP. O experimento mediu estes valores por cerca de um dia, segundo a segundo, e o cálculo de média foi realizado.

Para determinar as arquiteturas de melhor relação entre custo × benefício, precisamos primeiramente definir o que ou qual é o benefício que estamos buscando. Neste trabalho buscamos a relação ideal entre *downtime* e custo, mas, por sere grandezas diferentes, é necessário normalizá-los, e colocá-los dentro de um mesmo intervalo: 0, 1. A Expressão 4.2 apresenta o processo de normalização realizado.

$$NormalizarXY = \frac{NumX - MinNumX}{MaxNumX - MinNumX} \tag{4.2}$$

onde $X = Custo, Downtime$, $Y =$ Arquiteturas: A1, A2, A3, ..., A13, $MinNumX$ representa o menor valor para as arquiteturas e $MaxNumX$ representa o seu maior valor, já $NumX$ armazena o valor atual de custo ou *downtime* no momento de sua análise.

Agora de posse dos valores normalizados de custo e *downtime*, é preciso relacioná-los de alguma forma, utilizamos então da distância euclidiana e a arquitetura com menor distância da origem tende a ser a de melhor custo benefício. A Equação 4.3 mostra como é realizado o cálculo das distâncias.

$$DistanceZ = \sqrt{MN^2 + MN^2} \tag{4.3}$$

Tabela 4.13: Relação entre componente, potência e custos

Componente	Custo de Aquisição (US$)	Potência (W)	Custo Energético Unitário (US$)
Servidor	1.209,82	100	140,16
Switch	119,00	6,9	9,67
Ar Condicionado	490,83	1080	1.513,73
Monitor	191,95	70	98,112
Teclado	5,71	-	-
Mouse	2,55	-	-
Computador Pessoal	313,27	100	140,16
Rack	416,73	-	-

Tabela 4.14: Custos estimados para cada arquitetura

Arquitetura	**Nº de Máquinas**	**Custo Energético Anual (US$)**	**Custo de Aquisição (US$)**	**Custo Total (US$)**
A1	2	2.041,99	3.959,68	6.001,67
A2	3	2.182,15	5.169,50	7.351,65
A3	3	2.110,24	5.169,50	7.279,74
A4	3	2.047,24	5.169,50	7.216,74
A5	4	2.322,31	6.379,3	8.701,63
A6	4	2.250,40	6.379,3	8.629,72
A7	4	2.187,40	6.379,3	8.566,72
A8	4	2.250,40	6.379,3	8.629,72
A9	4	2.178,50	6.379,3	8.557,82
A10	4	2.115,50	6.379,3	8.494,82
A11	4	2.187,40	6.379,3	8.566,72
A12	4	2.115,50	6.379,3	8.494,82
A13	4	2.052,50	6.379,3	8.431,82

onde $Z = Arquiteturas : A1, A2, A3, ..., A13$, MN = Custo Normalizado, Downtime Normalizado, Disponibilidade Normalizada.

Relação entre Custo e Downtime

O Ranking para a relação entre custo e *downtime* é apresentado na Tabela 4.15.

Tabela 4.15: Ranking de Arquiteturas (Custo × Downtime)

Posição	Arquitetura	Custo Normalizado	Downtime Normalizado	Distância
1º	A4	0,45003	0,31974	0,52548
2º	A3	0,47337	0,29833	0,53027
3º	A2	0,50000	0,28667	0,53953
4º	A13	0,90007	0,07312	0,67284
5º	A10	0,92340	0,03597	0,67996
6º	A12	0,92340	0,05152	0,68046
7º	A9	0,94674	0,01444	0,68809
8º	A7	0,95003	0,03326	0,68962
9º	A11	0,95003	0,03986	0,68979
10º	A8	0,97337	0,01172	0,69763
11º	A6	0,97337	0,03597	0,69768
12º	A1	0,00000	1,00000	0,70711
12º	A5	1,00000	0,00000	0,70711

O ranking mostra que a arquitetura A3, com nó em redundância do tipo *warm standby* possui a melhor relação entre custo e *downtime*, o que também pode ser visto através da Figura 4.6. É importante ressaltar que esta arquitetura não possui a maior disponibilidade nem o pior custo, mas quando se relaciona fatores através do método de normalização e distância Euclidiana, caso atribuídos um mesmo peso é isto que obtemos. O último lugar é dividido entre duas arquiteturas, a com menor custo e a de maior disponibilidade, A1 e A5 respectivamente.

Neste tipo de gráfico, podemos visualizar as arquiteturas que apresentam a melhor relação entre custo, em laranja, e *downtime* em azul, estas que são respectivamente A4, A2 e A3, já que apesar de estarem próximas do ponto de origem do nosso radar, tanto para suas linhas azuis quanto para as alaranjadas, estão distantes de seus rótulos. A arquitetura que possui o maior *downtime* é a arquitetura A1, que atinge uma maior proximidade do rótulo através da linha azul, já a arquitetura A5 é a que possui o maior custo, sendo a linha alaranjada a mais próxima de seu rótulo, as arquiteturas A1 e A5 são corroboradas pela métrica contrária, possuindo respectivamente o menor custo e o menor *downtime*. Outra forma de se representar a relação entre custo e *downtime* é através do tradicional gráfico em linha (Figura 4.7) a distância euclidiana entre cada ponto que representa uma arquitetura à origem.

É possível atribuir pesos distintos às métricas de interesse, ou seja, dar maior importância a algumas das métricas de interesse do que a outras. A Tabela 4.16 apresenta uma relação onde o *downtime* é mais importante que o custo.

No segundo cenário analisamos o *downtime* com peso maior que o custo, esta relação apresentou a arquitetura A3 no topo do ranking, nesta arquitetura os nós estão em redundância

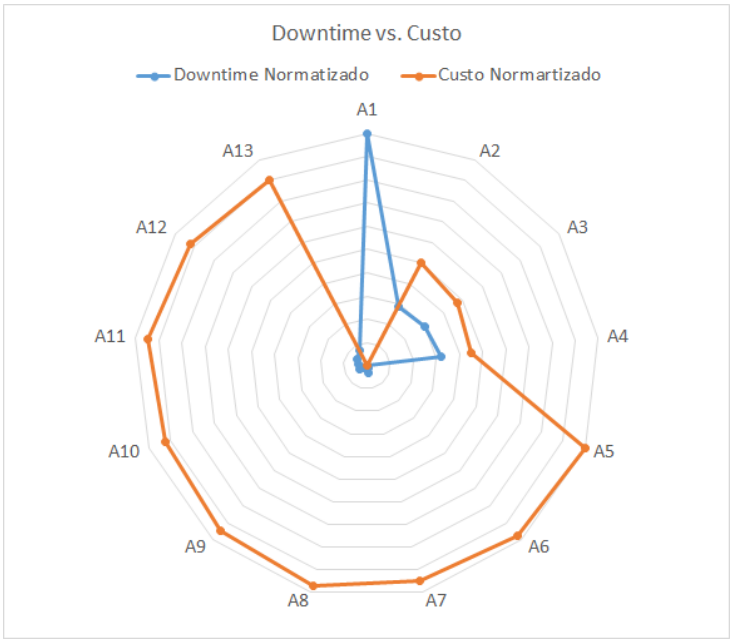


Figura 4.6: Comparativo entre Downtime e Custo (Radar)

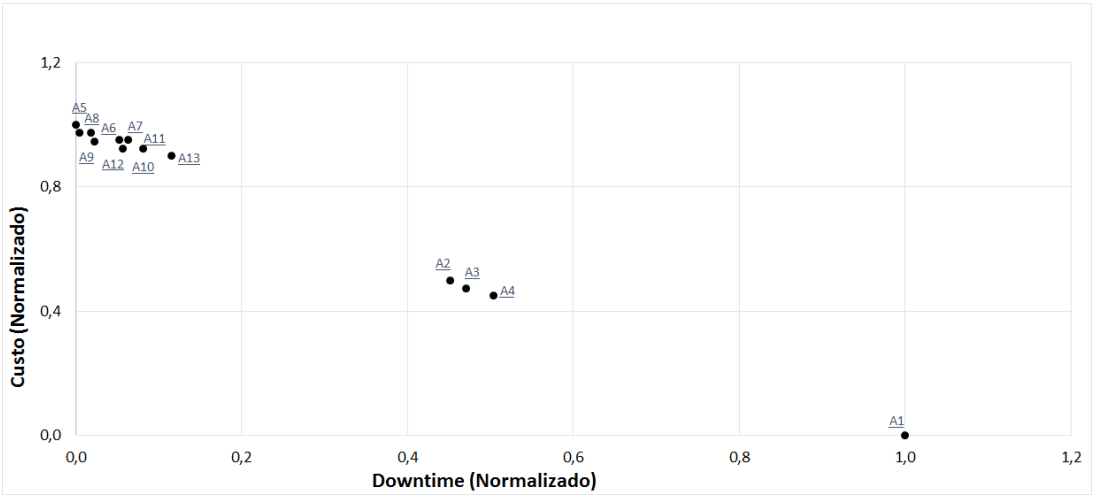


Figura 4.7: Comparativo entre Downtime e Custo (Linha)

Tabela 4.16: Ranking das arquiteturas com *downtime* 75% e custo 25%

Posição	Arquitetura	Downtime Normalizado	Custo Normalizado	Distância
1º	A3	0,29833	0,47337	0,43022
2º	A2	0,28667	0,50000	0,43201
3º	A4	0,31974	0,45003	0,43495
4º	A13	0,07312	0,90007	0,47857
5º	A10	0,03597	0,92340	0,48148
6º	A12	0,05152	0,92340	0,48254
7º	A9	0,01444	0,94674	0,48666
8º	A7	0,03326	0,95003	0,48820
9º	A11	0,03986	0,95003	0,48857
10º	A8	0,00271	0,97337	0,49330
11º	A6	0,01172	0,97337	0,49340
12º	A5	0.00000	1.00000	0,50000
13º	A1	1.00000	0.00000	0,86603

do tipo *warm standby*. Em último lugar o resultado já esperado, a arquitetura: A1 ou *baseline*. Já o próximo ranking apresenta a relação onde o custo é mais importante que o *downtime*, visto na Tabela 4.17.

Tabela 4.17: Ranking das arquiteturas com custo 75% e *downtime* 25%

Posição	Arquitetura	Downtime Normalizado	Custo Normalizado	Distância
1º	A1	1,00000	0,00000	0,50000
2º	A4	0,31974	0,45003	0,60252
3º	A3	0,29833	0,47337	0,61423
4º	A2	0,28667	0,50000	0,62892
5º	A13	0,07312	0,90007	0,82243
6º	A10	0,03597	0,92340	0,83239
7º	A12	0,05152	0,92340	0,83260
8º	A9	0,01444	0,94674	0,84268
9º	A7	0,03326	0,95003	0,84428
10º	A11	0,03986	0,95003	0,84435
11º	A8	0,00271	0,97337	0,85442
12º	A6	0,01172	0,97337	0,85443
13º	A5	0,00000	1,00000	0,86603

Em um cenário, onde o custo tem uma maior importância que o *downtime* a arquitetura A1 leva uma larga vantagem sobre as demais, já que com um menor número de servidores vem um menor custo total que é ainda menor por conta desta normalização adicionada de pesos. Na sequência, encontram-se A4, A3 e A2, ou seja, com redundância apenas nos nós, novamente o custo dos servidores é levado em consideração, e aqui apenas três são utilizados, diferentemente das demais arquiteturas. O pior desempenho é da arquitetura A5, que possui 4 servidores em operação.

Considerações

Este Capítulo apresentou os resultados provenientes deste trabalho, demonstrado o quão factível são os modelos propostos através da utilização de estudos de caso que cobriam cenários reais enfrentados por empresas que proveem infraestruturas e serviços de tecnologia e informação. Treze tipos de arquiteturas foram apresentados e valores de disponibilidade e custo para cada uma foi levantado, além de uma análise do real provimento do sistema e a quantas pessoas seria entregue com base em uma análise de disponibilidade orientada à capacidade. O próximo Capítulo apresenta nossas conclusões e propõe uma série de trabalhos futuros a serem realizados.

5

Conclusões e Trabalhos Futuros

Usuários de sistemas computacionais estão aderindo cada vez mais às tecnologias voltadas à sincronização de dados entre dispositivos. Sendo de grande importância aos usuários deste tipo de sistemas o acesso a dados pessoais já sincronizados de forma rápida, segura e a partir de qualquer ponto com acesso à Internet. O estudo aqui realizado, bem como conceitos, estudos de caso e modelos apresentados, podem, também, serem aplicados a serviços, sistemas e empreendimentos que dependam das tecnologias de sincronização de dados. É o caso dos sistemas bancários e comerciais que precisam permanecer em disponibilidade o máximo de tempo possível para evitar que transações de crédito e débito não sejam devidamente efetivadas devido a falta da sincronia entre suas bases de dados e sistemas.

Em geral, seja no âmbito acadêmico, pessoal ou comercial, as tecnologias destinadas à sincronização trabalham em cima de arquiteturas do tipo cliente-servidor e, as empresas que oferecem este tipo de serviço, há muito migraram ou estão migrando em grande escala para o paradigma de computação em nuvem, já que a possibilidade de recursos virtualmente ilimitados proporciona uma ótima opção para os usuários deste tipo de serviço, além de fatores relacionados à segurança de dados em infraestruturas de computação em nuvem pessoais e privadas, além de, é claro, o total controle sobre o seu ambiente caso o gestor opte pela implantação de sua própria infraestrutura de nuvem, algo que é garantido por uma série de ferramentas e *frameworks* voltados ao gerenciamento deste tipo de ambiente.

Este trabalho propôs uma série de arquiteturas que representam um servidor de sincronização de dados hospedado em plataforma privada de computação em nuvem, os custos para implantação de cada uma das arquiteturas é levantado, bem como os valores para métricas de dependabilidade de cada uma delas, buscando encontrar a melhor relação entre custo $\times$ benefício. Uma análise de disponibilidade orientada à capacidade também é realizada, e em conjunto com uma técnica para análise de sensibilidade foi capaz de diagnosticar que o nó é o componente que mais impacta na real disponibilidade dos recursos/máquinas virtuais da infraestrutura.

Uma série de modelos baseados em estados como RBDs, CTMCs, e SPNs foram desenvolvidos através da ferramenta Mercury, estes modelos foram dispostos de maneira hierárquica, visando à realização das avaliações de dois estudos de caso que se basearam na análise de

disponibilidade para as arquiteturas propostas, mostrando o quão factível eles eram de acordo com os resultados obtidos.

A disponibilidade de um total de 13 arquiteturas foi avaliada, 12 delas eram variantes de uma arquitetura básica que contém os componentes mínimos necessários para o fornecimento do serviço de sincronização dados em uma plataforma gerenciada pelo Eucalyptus: um frontend e um nó. As variações foram decorrentes da aplicação da técnica de diferença percentual, que destina-se à análise de sensibilidade em sistemas.

Técnicas de análise de sensibilidade são utilizadas quando buscamos os componentes que apresentam o maior impacto em uma métrica de interesse do sistema, que no nosso caso, tratava-se da disponibilidade, a arquitetura básica apresentou uma disponibilidade de 98,82%, o que aponta para um período de *downtime* de aproximadamente 103 horas no decorrer de um ano, a análise de sensibilidade apontou que os maiores responsáveis pela indisponibilidade do sistema eram os tempos para falha e reparo do banco de dados utilizado pelo servidor de sincronização e em execução na máquina virtual gerida pelo hypervisor no nó da arquitetura básica.

Com base nos resultados obtidos pela análise de sensibilidade na arquitetura básica, a criação de três novas arquiteturas com a aplicação de três técnicas de redundância foi realizada no nó que hospedava o serviço, a técnica de destaque foi a *hot standby* que conseguiu reduzir o *downtime* anual do sistema para 46,74 horas, atingindo uma disponibilidade de 99,46%. Por ainda considerar este valor inferior ao que almejamos, uma nova análise de sensibilidade foi aplicada à arquitetura com os dois nós em *hot standby*, esta análise apontou que os componentes do frontend passaram a ter maior impacto na disponibilidade do sistema. Um novo conjunto de 9 arquiteturas contendo técnicas de redundância aplicadas ao nó e ao frontend puderam elevar a disponibilidade do sistema, que para o caso da aplicação de redundâncias do tipo *hot standby* ao frontend e ao nó proporcionou uma disponibilidade de 99,996%, e um *downtime* anual de 0,28 horas.

É importante salientar que uma etapa de validação da aplicação responsável pela sincronia efetiva dos dados também foi realizada, esta etapa ocorreu paralelamente a criação dos modelos de disponibilidade, e consistiu na realização de experimentos em uma infraestrutura que representava a arquitetura básica. O experimento realizado teve como base a injeção e monitoração de falhas no sincronizador, via *scripts bash*, os resultados comprovaram que os valores escolhidos para falha e reparo do sincronizador encontram-se dentro de um intervalo de confiança de 95%, o que demonstra a factividade dos valores escolhidos e gerados pelos modelos de disponibilidade.

Um modelo para análise de disponibilidade orientada à capacidade também foi apresentado, neste modelo a real capacidade do sistema para fornecer as máquinas virtuais que hospedam o serviço de sincronização de dados proposto foi investigada, e avaliou um número de máquinas virtuais/recursos que é perdido com base na ocorrência de falhas, relacionando a disponibilidade de uma arquitetura com nós em redundância *hot standby* ao número máximo de máquinas suportados por cada nó da arquitetura.

Já no segundo estudo de caso, onde uma análise de custos de implantação foi realizada,

o levantamento dos preços de aquisição e de consumo energético médio para cada um dos componentes pertencentes e responsáveis pelo funcionamento das arquiteturas propostas foi realizado. O objetivo deste levantamento foi diagnosticar qual/quais das 13 arquiteturas propostas apresentavam uma melhor relação entre custos e *downtime*. A utilização de técnicas para normalização de grandezas e cálculo de distância Euclidiana comprovaram que a arquitetura com nó redundante em *warm standby* possui a melhor relação, enquanto que a arquitetura básica apresenta os piores resultados, em uma avaliação onde custo e *downtime* possuem a mesma importância.

As principais contribuições desta dissertação foram os artefatos, compostos por modelos baseados estocásticos que representam o comportamento do sistema, e pelas avaliações de disponibilidade, disponibilidade orientada à capacidade, custos de implantação, e melhor relação entre custo e benefício para infraestruturas computacionais, que são direcionadas a empresas que oferecem, ou que visam oferecer, serviços para a sincronização de dados.

Quanto à possíveis trabalhos futuros, podemos listar além das análises e avaliações de outras métricas de dependabilidade, como a confiabilidade, aplicadas ao serviço de sincronização de dados também pelo lado do cliente, ou seja, aplicações móveis para sincronização de dados e como estas estão interagindo com o servidor hospedado em uma plataforma de computação em nuvem. O desempenho da sincronização com relação a tempos e envio de dados também é de grande importância, já que associados as métricas de dependabilidade podem prover ao administrador de uma infraestrutura ou serviço de sincronização valores de performabilidade que garantem a quem contrata ou utiliza o serviço, um retorno ainda mais seguro e confiável daquilo em que estão investindo seus recursos financeiros.

Referências

AGARWAL, S.; STAROBINSKI, D.; TRACHTENBERG, A. On the scalability of data synchronization protocols for PDAs and mobile devices. **Network, IEEE**, [S.l.], v.16, n.4, p.22–28, Jul 2002.

AGERWALA, T. Special Feature: putting petri nets to work. **Computer**, Los Alamitos, CA, USA, v.12, n.12, p.85–94, 1979.

ARAUJO, J. et al. Experimental Evaluation of Software Aging Effects on the Eucalyptus Cloud Computing Infrastructure. In: MIDDLEWARE 2011 INDUSTRY TRACK WORKSHOP, New York, NY, USA. **Proceedings...** ACM, 2011. p.4:1–4:7. (Middleware '11).

AVIŽIENIS, A. et al. **Fundamental Concepts of Dependability**. [S.l.]: University of Newcastle upon Tyne, Computing Science, 2001. (Technical report series).

AVIZIENIS, A. et al. Basic Concepts and Taxonomy of Dependable and Secure Computing. **IEEE Transactions on Dependable and Secure Computing**, [S.l.], v.1, p.11–33, 2004.

BRINKSMA, E.; HERMANNS, H.; KATOEN, J.-P. (Ed.). **Lectures on Formal Methods and PerformanceAnalysis**: first eef/euro summer school on trends in computer science bergen dal, the netherlands, july 3–7, 2000 revised lectures. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. p.84–155.

BEZERRA, M. C. et al. Availability modeling and analysis of a VoD service for eucalyptus platform. In: SYSTEMS, MAN AND CYBERNETICS (SMC), 2014 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014. p.3779–3784.

BOLCH, G. et al. Modeling and Performance Evaluation with Computer Science Application. In: QUEUING NETWORKS AND MARKOV CHAINS. **Anais...** [S.l.: s.n.], 2006.

BOREN, Z. D. **There are officially more mobile devices than people in the world**. [Online; accessed 19-December-2015], `http://goo.gl/NJGc7E`.

CALLOU, G. et al. Models for dependability and sustainability analysis of data center cooling architectures. In: DEPENDABLE SYSTEMS AND NETWORKS WORKSHOPS (DSN-W), 2012 IEEE/IFIP 42ND INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2012. p.1–6.

CAMPOS, E. et al. Stochastic Modeling of Auto Scaling Mechanism in Private Clouds for Supporting Performance Tunning. In: IEEE INT. CONF. ON SYSTEMS, MAN, AND CYBERNETICS (SMC'15), Hong Kong, China. **Proceedings...** [S.l.: s.n.], 2015.

CASSANDRAS, C.; LAFORTUNE, S. **Introduction to Discrete Event Systems**. [S.l.]: Kluwer Academic, 1999. (Discrete event dynamic systems).

CHUN-LEI, C.; JIAN-QIANG, D. Design and implementation of data synchronization in embedded TCM system based on SyncML. In: COMPUTER APPLICATION AND SYSTEM MODELING (ICCASM), 2010 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. v.10, p.V10–619–V10–622.

CHUN-LEI, C.; JIANG-QING, D. Design and Implementation of data synchronization in Embedded TCM System Based on SyncML. **2010 International Conference on Computer Application and System Modeling**, Taiyuan, v.10, p.619–622, 2010.

COSTA, I. et al. Availability Evaluation and Sensitivity Analysis of a Mobile Backend-as-a-Service Platform. **Journal Quality and Reliability Engineering International**, [S.l.], 2015.

DANTAS, J. **Modelos para Analise de Dependabilidade de Arquiteturas de Computação em Nuvem**. 2013. Dissertação (Mestrado em Ciência da Computação) — Centro de Informática - Universidade Federal de Pernambuco (Recife, Brasil).

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

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

DANTAS, J. et al. Eucalyptus-based private clouds: availability modeling and comparison to the cost of a public cloud. **Computing**, [S.l.], v.97, n.11, p.1121–1140, 2015.

EJLALI, A. et al. A Hybrid Fault Injection Approach Based on Simulation and Emulation Co-operation. **2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks**, Los Alamitos, CA, USA, v.0, p.479, 2003.

ESTEVAN COSTA, F. M.; BRANCHER, J. Analise Comparativa da Fatoracaao LU: estudo de caso opencl cpu x gpu. **11vo Simposio Argentino de Investigacion Operativa, SIO 2013**, [S.l.], 2013.

FERREIRA, A. **Dicionário Aurélio da Lingua Portuguesa**. Brazil: Editora Positivo., 2014.

FERREIRA, J. et al. An Algorithm to Optimize Electrical Flows of Private Cloud Infrastructures. In: SYSTEMS, MAN, AND CYBERNETICS (SMC), 2015 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2015. p.771–776.

FOSTER, I. et al. Cloud Computing and Grid Computing 360-Degree Compared. In: GRID COMPUTING ENVIRONMENTS WORKSHOP, Austin. **Anais...** [S.l.: s.n.], 2008.

FRANK, P. M. **Introduction to Sensitivity Analysis**. [S.l.]: Academic, 1978.

FUNAMBOL. **Funambol version 8.5 Installation and Administration Guide**. [Online; accessed 21-December-2015].

GARG, S. et al. Analysis of software rejuvenation using Markov regenerative stochastic Petri net. In: IN: SIXTH INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING, (ISSRE'95), Paderborn. **Proceedings...** [S.l.: s.n.], 1995. p.180–187.

GERMAN, R. **Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets**. New York, NY, USA: John Wiley & Sons, Inc., 2000.

HAUCK, M. et al. **Challenges and Opportunities of Cloud Computing**. [S.l.]: UC Berkeley Reliable Adaptive Distributed Systems Laboratory, 2009. (UCB/EECS-2009-28).

HERZOG, U. Lectures on Formal Methods and Performance Analysis. In: BRINKSMA, E.; HERMANNS, H.; KATOEN, J.-P. (Ed.). . New York, NY, USA: Springer-Verlag New York, Inc., 2002. p.1–37.

HORDE. **SyncML (Synchronization Markup Language)**. 2013.

HP. **Virtual Machine Types**. [Online; accessed 19-December-2015], `https://docs.hpcloud.com/eucalyptus/4.2.2/user-guide/vm_types.html`.

IBM. **IBM SmartCloud**. [Online; Acesso em 15-Agosto-2014], `http://www.ibm.com/cloud-computing/br/pt/`.

IEEE. IEEE Standard Glossary of Software Engineering Terminology. In: IEEE STD. **Anais...** IEEE Computer Society, 1990.

JAIN, R. **The Art of Computer Systems Performance Analysis**: techniques for experimental design, measurement, simulation, and modeling. New York: Wiley Computer Publishing, John Wiley & Sons, Inc., 1991.

JOHNSON et al. **Eucalyptus Beginner's Guide**. UEC.ed. [S.l.: s.n.], 2010. For Ubuntu Server 10.04 - Lucid Lynx, v1.0.

KEESE, W. R. A Method of determining a Confidence Interval for Availability. **Reliability Office**, Point Mugu, CA, USA, p.10, 1965.

KUO, W.; ZUO, M. **Optimal Reliability Modeling**: principles and applications. [S.l.]: Wiley, 2003.

LEE, B. Y.; LEE, G. H.; KIM, Y. S. Modeling and Analysis of SyncML Server System Using Stochastic Petri Nets. In: WIRELESS AND MOBILE COMMUNICATIONS, 2007. ICWMC '07. THIRD INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2007. p.60–60.

LI, J.-L.; LI, J.-P. Data synchronization protocol in mobile computing environment using SyncML and Huffman coding. In: WAVELET ACTIVE MEDIA TECHNOLOGY AND INFORMATION PROCESSING (ICWAMTIP), 2012 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2012. p.260–262.

MACIEL, P. et al. Dependability Modeling. In: PERFORMANCE AND DEPENDABILITY IN SERVICE COMPUTING: CONCEPTS, TECHNIQUES AND RESEARCH DIRECTIONS. **Anais...** [S.l.: s.n.], 2011.

MACIEL, P.; LINS, R.; CUNHA, P. **Uma Introducao as Redes de Petri e Aplicacoes**. [S.l.]: Sociedade Brasileira de Computacao, 1996. 213p.

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

MAO, M.; CHEN, S.; XU, J. Construction of the Initial Structure for Preimage Attack of MD5. In: COMPUTATIONAL INTELLIGENCE AND SECURITY, 2009. CIS '09. INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2009. v.1, p.442–445.

MARSAN, M. A. et al. **Modelling with Generalized Stochastic Petri Nets**. 1st.ed. New York, NY, USA: John Wiley & Sons, Inc., 1994.

MELO, R. M. D. et al. Redundant VoD Streaming Service in a Private Cloud: availability modeling and sensitivity analysis. **Mathematical Problems in Engineering**, [S.l.], v.2014, p.14, 2014.

MOLLOY, M. K. **On the Integration of Delay and Throughput Measures in Distributed Processing Models**. 1981. Tese (Doutorado em Ciência da Computação) — . AAI8201138.

MONTGOMERY, D. C. **Design and Analysis of Experiments**. [S.l.]: John Wiley & Sons, 2006.

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

NIRIX. **CLOUD SOLUTIONS FOR BUSINESSES**. Online; accessed 21-January-2015, `http://www.nirix.com/cloud-solutions-for-businesses/`.

OMA, O. M. A. **Device Management**. 2012.

PETERSON, W. W.; BROWN, D. T. Cyclic Codes for Error Detection. **Proceedings of the IRE**, [S.l.], v.49, n.1, p.228–235, Jan 1961.

RACKSPACE. **Rackspace the open cloud company**. Online; accessed 21-December-2013, `http://www.rackspace.com/pt/`.

REGALADO, A. **Who Coined Cloud Computing?** [Online; accessed 21-December-2013], `http://www.technologyreview.com/news/425970/who-coined-cloud-computing/`.

SA, T. T.; SOARES, J. M.; GOMES, D. G. CloudReports: uma ferramenta gráfica para a simulação de ambientes computacionais em nuvem baseada no framework cloudsim. **2011 IX Workshop em Clouds e Aplicações - WCGA**, [S.l.], 2011.

SAITO, Y.; SHAPIRO, M. Optimistic Replication. **ACM Computing Surveys**, New York, v.37, p.42–81, 2005.

SILVA, B. et al. Mercury: an integrated environment for performance and dependability evaluation of general systems. In: INDUSTRIAL TRACK AT 45TH DEPENDABLE SYSTEMS AND NETWORKS CONFERENCE (DSN-2015), Rio de Janeiro, Brazil. **Proceedings...** [S.l.: s.n.], 2015.

SMOLAREK, T. **Study of Open Mobile Alliance Device Management sessions for most effective device management.** 2011. Dissertação (Mestrado em Ciência da Computação) — School of Computing - Blekinge Institute of Technology (Karlskrona, Sweden).

SOUSA, E. et al. Performance modeling for evaluation and planning of Electronic Funds Transfer Systems. In: ISCC'09. **Anais...** [S.l.: s.n.], 2009. p.73–76.

SOUSA, E. et al. Dependability evaluation of cloud infrastructures. In: SYSTEMS, MAN AND CYBERNETICS (SMC), 2014 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014. p.1282–1287.

SUN. **Introduction to Cloud Computing Architecture**. 1.ed. [S.l.]: Sun Microsystems, Inc., 2009.

TIAN, Y.; LI, J.-P. Research and implementation of data synchronization with SyncML. **2012 9th International Computer Conference on Wavelet Active Media Technology and Information Processing**, Chengdu, p.302–304, 2012.

TRAUD, A. et al. Cyclic Data Synchronization Through Reusing SyncML. **2008 The Ninth International Conference on Mobile Data Management**, Beijing, p.165–172, 2008.

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

VAQUEIRO, L. M. et al. A Break in the Clouds: towards a cloud definition. **Computer Communication Review**, [S.l.], v.39, p.50–55, 2009.

YANG, R. et al. D2PS: a dependable data provisioning service in multi-tenant cloud environment. In: IEEE 17TH INTERNATIONAL SYMPOSIUM ON HIGH ASSURANCE SYSTEMS ENGINEERING (HASE), 2016. **Anais...** [S.l.: s.n.], 2016. p.252–259.

Apêndice

A

Mercury

O Mercury é uma ferramenta desenvolvida pelo grupo de pesquisa Modeling of Distributed and Concurrent Systems (MoDCS) no Centro de Informática da Universidade Federal de Pernambuco, e que permite a criação e avaliação dos mais diversos modelos de desempenho e dependabilidade SILVA et al. (2015), tendo papel importantíssimo no desenvolvimento desta pesquisa de mestrado.

Com este poderoso ambiente integrado de desenvolvimento e modelagem gráfica é possível representar os seguintes formalismos: CTMC, SPN, RBD, Energy Flow Models (EFM) e, através de técnicas de análise de sensibilidade implementadas na ferramenta, é possível determinar os componentes que detenham maior importância dentro dos modelos de sistemas criados.

A Figura A.1 apresenta o exemplo de uma SPN no ambiente gráfico da ferramenta Mercury.

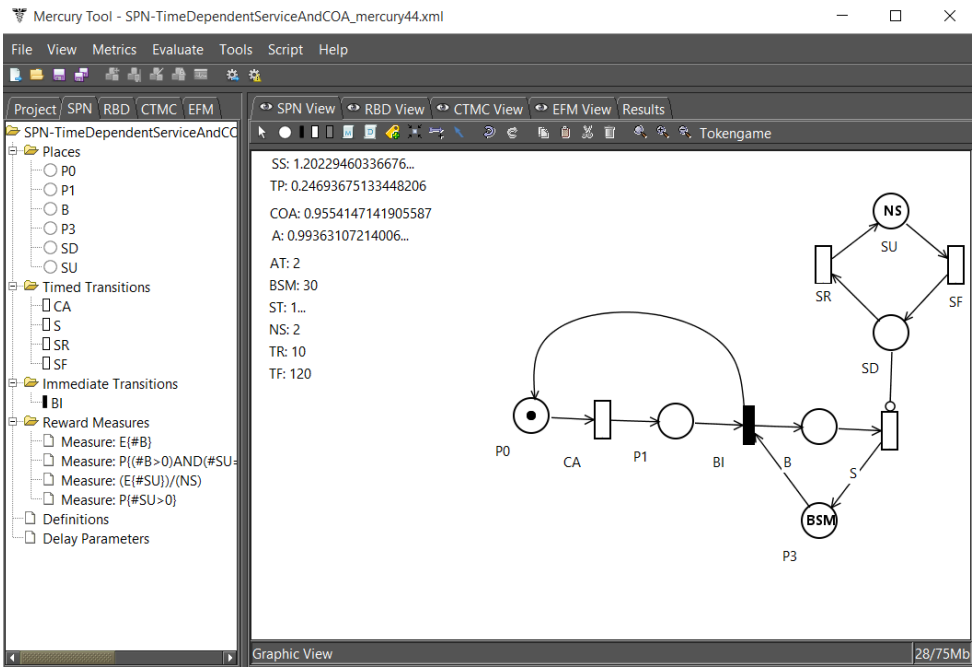


Figura A.1: Exemplo de uma SPN na ferramenta Mercury

Já por vias textuais o Mercury proporciona um ambiente para escrita, avaliação e modelagem hierárquica de redes de Petri, cadeias de Markov e diagramas de bloco de confiabilidade, e foi através dessa linguagem que os modelos apresentados nesta dissertação foram avaliados, os *scripts* que resultaram em nossos modelos são apresentados nos capítulos à seguir, com início pela avaliação de disponibilidade e sequenciados pela análises de disponibilidade orientada à capacidade e de sensibilidade.

B

Scripts para Avaliação de Disponibilidade dos subsistemas

Listing B.1: Script Mercury para avaliação de disponibilidade do frontend

```
RBD frontend{

block HW( MTTF = mttf_hw, MTTR = mttr_hw);
block OS( MTTF = mttf_os, MTTR = mttr_os);
block CC( MTTF = mttf_cc, MTTR = mttr_cc);
block CLC( MTTF = mttf_clc, MTTR = mttr_clc);
block SC( MTTF = mttf_sc, MTTR = mttr_sc);
block Walrus( MTTF = mttf_walrus, MTTR = mttr_walrus);
series s0(HW, OS, CC, CLC, SC, Walrus );

top s0;

metric av = availability;
}
```

Listing B.2: Script Mercury para avaliação de disponibilidade do nó

```
RBD node{

block HW( MTTF = mttf_hw_no, MTTR = mttr_hw_no);
block OS( MTTF = mttf_os_no, MTTR = mttr_os_no);
block Hypervisor( MTTF = mttf_hypervisor, MTTR = mttr_hypervisor);
block NC( MTTF = mttf_nc, MTTR = mttr_nc);
series s0(HW, OS, Hypervisor, NC );

top s0;

metric av = availability;
}
```

Listing B.3: Script Mercury para avaliação de disponibilidade do serviço de sincronização

```
CTMC service{
```

```
state UUUU up;
state UUUD;
state UUDD;
state UDUD;
state UDDD;
state UUDU;
state UDUU;
state UDDU;
state DDDD;

transition UUUU -> UUUD( rate = lambdasyncml);
transition UUUU -> UUDU( rate = lambdaapache);
transition UUUU -> UDUU( rate = lambdadb);
transition UUUU -> DDDD( rate = lambdavm);
transition UUUD -> UUUU( rate = musyncml);
transition UUUD -> UUDD( rate = lambdaapache);
transition UUUD -> UDUD( rate = lambdadb);
transition UUUD -> DDDD( rate = lambdavm);
transition UUDD -> UUUD( rate = muapache);
transition UUDD -> UDDD( rate = lambdadb);
transition UUDD -> UUDU( rate = musyncml);
transition UUDD -> DDDD( rate = lambdavm);
transition UDUD -> UUUD( rate = mudb);
transition UDUD -> UDDD( rate = lambdaapache);
transition UDUD -> UDUU( rate = musyncml);
transition UDUD -> DDDD( rate = lambdavm);
transition UDDD -> UUUU( rate = reboot);
transition UDDD -> DDDD( rate = lambdavm);
transition UUDU -> UUUU( rate = muapache);
transition UUDU -> UUDD( rate = lambdasyncml);
transition UUDU -> UDDU( rate = lambdadb);
transition UUDU -> DDDD( rate = lambdavm);
transition UDUU -> UUUU( rate = mudb);
transition UDUU -> UDUD( rate = lambdasyncml);
transition UDUU -> UDDU( rate = lambdaapache);
transition UDUU -> DDDD( rate = lambdavm);
transition UDDU -> UDDD( rate = lambdasyncml);
transition UDDU -> UUDU( rate = mudb);
transition UDDU -> UDUU( rate = muapache);
transition UDDU -> DDDD( rate = lambdavm);
transition DDDD -> UUUU( rate = reboot);

metric av = availability ;
metric av2 = stationaryAnalysis( expression = "P{UUUU}" );
}
```

Listing B.4: Script Mercury para avaliação de disponibilidade do nó *warm standby*

```
CTMC service{

state UWU up;
state UDU up;
state DUW;
state UUW up;
state DDD;
state DUD;
state UUD up;
state DDW;
state DDU;
```

```
state DWU;
state DWD;

transition UWU -> UDU( rate = lambdawnc);
transition UWU -> DWU( rate = lambdaservice);
transition UWU -> DWD( rate = lambdanc);
transition UDU -> UWU( rate = munc);
transition UDU -> DDD( rate = lambdanc);
transition UDU -> DDU( rate = lambdaservice);
transition DUW -> UUW( rate = muservice);
transition DUW -> DUD( rate = lambdawnc);
transition DUW -> DDW( rate = lambdanc);
transition UUW -> DUW( rate = lambdaservice);
transition UUW -> UUD( rate = lambdawnc);
transition UUW -> DDW( rate = lambdanc);
transition DDD -> DUD( rate = munc);
transition DDD -> DDU( rate = munc);
transition DUD -> DUW( rate = munc);
transition DUD -> DDD( rate = lambdanc);
transition DUD -> UUD( rate = muservice);
transition UUD -> UUW( rate = munc);
transition UUD -> DDD( rate = lambdanc);
transition UUD -> DUD( rate = lambdaservice);
transition DDW -> DUW( rate = munc);
transition DDW -> DDD( rate = lambdawnc);
transition DDW -> DDU( rate = sa);
transition DDU -> UDU( rate = muservice);
transition DDU -> DDD( rate = lambdanc);
transition DDU -> DWU( rate = munc);
transition DWU -> UWU( rate = muservice);
transition DWU -> DDU( rate = lambdawnc);
transition DWU -> DWD( rate = lambdanc);
transition DWD -> DDD( rate = lambdawnc);
transition DWD -> DUD( rate = sa);
transition DWD -> DWU( rate = munc);

metric av = availability ;
metric av2 = stationaryAnalysis( expression = "P{UWU}+P{UDU}P{UUW}P{UUD}" );
}
```

Listing B.5: Script Mercury para avaliação de disponibilidade do componente em *cold standby*

```
SPN componente{

place ComponentePrincipaUP( tokens= 1 );
place ComponentePrincipalAguarda;
place ComponentePrincipalDOWN;
place ComponenteRedundanteDOWN;
place ComponenteRedundanteUP;
place ComponenteReservaAguarda( tokens= 1 );


immediateTransition ComponentePrincipalAtivo(
inputs = [ComponenteRedundanteUP, ComponentePrincipalAguarda],
outputs = [ComponenteReservaAguarda],
inhibitors = [ComponentePrincipalDOWN]
);
```

```
timedTransition ComponenteReservaAtivo(
inputs = [ComponenteReservaAguarda],
outputs = [ComponentePrincipalAguarda, ComponenteRedundanteUP],
inhibitors = [ComponentePrincipaUP],
delay = tempodechaveamento
);

timedTransition MTTFPrincipal(
inputs = [ComponentePrincipaUP],
outputs = [ComponentePrincipalDOWN],
delay = tempodefalha
);

timedTransition MTTFRedudante(
inputs = [ComponenteRedundanteUP],
outputs = [ComponenteRedundanteDOWN],
delay = tempodefalha+20%
);

timedTransition MTTRPrincipal(
inputs = [ComponentePrincipalDOWN],
outputs = [ComponentePrincipaUP],
delay = tempodereparo
);

timedTransition MTTRRedundante(
inputs = [ComponenteRedundanteDOWN],
outputs = [ComponenteRedundanteUP],
delay = tempodereparo
);

metric m1 = stationaryAnalysis ( expression = "P{(#ComponentePrincipaUP = 1) OR (#
    ComponenteRedundanteUP = 1)}" ) ;

}
```

C

Scripts para Avaliação de Disponibilidade orientada a capacidade

Listing C.1: Script Mercury para avaliação de disponibilidade orientada a capacidade

```
MTTF_VM = 2880;
MTTR_VM = 0.019;
MTTF_NODE = 484.81;
MTTR_NODE = 0.91;
NMVM = 12;


SPN Model{

place S_F;
place S_UP( tokens= 1 );
place VM_F;
place VM_UP( tokens= 12 );


immediateTransition VMF2(
enablingFunction = "#VM_UP>0",
inputs = [VM_UP(12)],
outputs = [VM_F(12)],
inhibitors = [S_UP]
);

timedTransition SF(
inputs = [S_UP],
outputs = [S_F],
delay = MTTF_NODE
);

timedTransition SR(
inputs = [S_F],
outputs = [S_UP],
delay = MTTR_NODE
);

timedTransition VMF(
inputs = [VM_UP],
```

```
outputs = [VM_F],
delay = MTTF_VM,
serverType = "InfiniteServer"
);

timedTransition VMR(
inputs = [VM_F],
outputs = [VM_UP],
inhibitors = [S_F],
delay = MTTR_VM
);

metric A = stationaryAnalysis( expression = "P{#VM_UP>0}" );
metric DTym = stationaryAnalysis( expression = "P{#VM_UP=0}*8760*60" );
metric COA = stationaryAnalysis( expression = "((P{#VM_UP=12}*12)+(P{#VM_UP=11}*11)+(P{#VM_UP
    =10}*10)+(P{#VM_UP=9}*9)+(P{#VM_UP=8}*8)+(P{#VM_UP=7}*7)+(P{#VM_UP=6}*6)+(P{#VM_UP=5}*5)+(
    P{#VM_UP=4}*4)+(P{#VM_UP=3}*3)+(P{#VM_UP=2}*2)+P{#VM_UP=1})/NMVM" );
}

main {
A = solve( Model,A );
println(A);

DTym = solve( Model,DTym );
println(DTym);

COA = solve( Model,COA );
println(COA);

}
```

D

Scripts para Avaliação de Disponibilidade das arquiteturas

Listing D.1: Script Mercury para avaliação de disponibilidade da arquitetura A1

```
RBD A1 {
hierarchy frontend (availability = solve ( frontend , av ));
hierarchy node (availability = solve ( node , av ));
hierarchy service (availability = solve ( service , av ));
series s1 (frontend ,node ,service);
top s1;
metric av = availability ;
metric uav ( 1 - av ) ;
metric downtime ( uav * 365 * 24 );
}
```

Listing D.2: Script Mercury para avaliação de disponibilidade da arquitetura A2

```
RBD A2{

block frontend( MTTF = mttf_frontend , MTTR = mttr_frontend);
block node1( MTTF = mttf_node1 , MTTR = mttr_node1);
block node2( MTTF = mttf_node2 , MTTR = mttr_node2);
parallel s0(node1,node2);
series s1(frontend , s0);

top s1;

top A2;
metric av = availability ;
metric uav ( 1 - av ) ;
metric downtime ( uav * 365 * 24 );
}
```

Listing D.3: Script Mercury para avaliação de disponibilidade da arquitetura A3

```
RBD A3 {
hierarchy frontend (availability = solve ( frontend , av ));
hierarchy service (availability = solve ( service , av ));
series s1 (frontend , service);
```

```
top s1;
metric av = availability ;
metric uav ( 1 - av ) ;
metric downtime ( uav * 365 * 24 );
}
```

Listing D.4: Script Mercury para avaliação de disponibilidade da arquitetura A4

```
RBD A4 {
hierarchy frontend (availability = solve ( frontend , av ));
hierarchy node (availability = solve ( node , ml ));
series s1 (frontend , node);
top s1;
metric av = availability ;
metric uav ( 1 - av ) ;
metric downtime ( uav * 365 * 24 );
}
```

Listing D.5: Script Mercury para avaliação de disponibilidade da arquitetura A5

```
RBD A5 {
hierarchy frontend (availability = solve ( frontend , av ));
hierarchy node (availability = solve ( node , av ));
parallel s0 (node, node);
parallel s1 (frontend, frontend);
series s2 (s0, s1);
top s2;
metric av = availability ;
metric uav ( 1 - av ) ;
metric downtime ( uav * 365 * 24 );
}
```

Listing D.6: Script Mercury para avaliação de disponibilidade da arquitetura A6

```
RBD A6 {
hierarchy frontend (availability = solve ( frontend , av ));
hierarchy service (availability = solve ( service , av ));
parallel s0 (frontend, frontend);
series s1 (s0 , service);
top s1;
metric av = availability ;
metric uav ( 1 - av ) ;
metric downtime ( uav * 365 * 24 );
}
```

Listing D.7: Script Mercury para avaliação de disponibilidade da arquitetura A7

```
RBD A7 {
hierarchy frontend (availability = solve ( frontend , av ));
hierarchy node (availability = solve ( node , ml ));
parallel s0 (frontend,frontend);
series s1 (s0 , node);
top s1;
```

```
metric av = availability ;
metric uav ( 1 - av ) ;
metric downtime ( uav * 365 * 24 );
}
```

Listing D.8: Script Mercury para avaliação de disponibilidade da arquitetura A8

```
RBD A8 {
hierarchy frontend (availability = solve ( frontend , av ));
hierarchy node (availability = solve ( node , av ));
parallel s0 (node, node);
series s1 (frontend, s0);
top s1;
metric av = availability ;
metric uav ( 1 - av ) ;
metric downtime ( uav * 365 * 24 );
}
```

Listing D.9: Script Mercury para avaliação de disponibilidade da arquitetura A9

```
RBD A9 {
hierarchy frontend (availability = solve ( frontend , av ));
hierarchy service (availability = solve ( service , av ));
series s1 (frontend , service);
top s1;
metric av = availability ;
metric uav ( 1 - av ) ;
metric downtime ( uav * 365 * 24 );
}
```

Listing D.10: Script Mercury para avaliação de disponibilidade da arquitetura A10

```
RBD A10 {
hierarchy frontend (availability = solve ( frontend , av ));
hierarchy node (availability = solve ( node , m1 ));
series s1 (frontend , node);
top s1;
metric av = availability ;
metric uav ( 1 - av ) ;
metric downtime ( uav * 365 * 24 );
}
```

Listing D.11: Script Mercury para avaliação de disponibilidade da arquitetura A11

```
RBD A11 {
hierarchy frontend (availability = solve ( frontend , m1 ));
hierarchy node (availability = solve ( node , av ));
parallel s0 (node, node);
series s1 (frontend, s0);
top s1;
metric av = availability ;
metric uav ( 1 - av ) ;
metric downtime ( uav * 365 * 24 );
}
```

Listing D.12: Script Mercury para avaliação de disponibilidade da arquitetura A12

```
RBD A12 {
hierarchy frontend (availability = solve ( frontend , ml ));
hierarchy service (availability = solve ( service , av ));
series s1 (frontend , service);
top s1;
metric av = availability ;
metric uav ( 1 - av ) ;
metric downtime ( uav * 365 * 24 );
}
```

Listing D.13: Script Mercury para avaliação de disponibilidade da arquitetura A13

```
RBD A13 {
hierarchy frontend (availability = solve ( frontend , ml ));
hierarchy node (availability = solve ( node , av ));
series s1 (frontend , node);
top s1;
metric av = availability ;
metric uav ( 1 - av ) ;
metric downtime ( uav * 365 * 24 );
}
```

E

Scripts para Análise de Sensibilidade

Listing E.1: Script Mercury para análise de sensibilidade da arquitetura A1

```
percentageDifference (
model_ = "A1",
metric_ = "av",
samplingPoints = 10,
parameters = (
lambdasyncml = [ 1/394.2, 1/1182.6 ] ,
musyncml = [ 1/0.5 , 1/1.5] ,
lambdaapache = [ 1/394.2 , 1/1182.6 ],
muapache = [ 1/0.5 , 1/1.5] ,
lambdadb = [ 1/720 , 1/2160],
mudb = [ 1/6.5 , 1/19.5 ],
lambdavm = [ 1/1440 , 1/4320 ],
reboot = [ 1/0.009583 , 1/0.028749 ],
mttf_hw = [ 4380, 13140 ],
mttr_hw = [ 0.5 , 1.5 ] ,
mttf_os = [ 1446.5 , 4339.5 ] ,
mttr_os = [ 0.125, 0.375 ],
mttf_hw_no = [ 4380, 13140 ],
mttr_hw_no = [ 0.5 , 1.5 ] ,
mttf_os_no = [ 1446.5 , 4339.5 ] ,
mttr_os_no = [ 0.125, 0.375 ],
mttf_hypervisor = [ 1495 , 4485 ],
mttr_hypervisor = [ 0.5, 1.5 ],
mttf_sc = [ 394.2 , 1182.6 ],
mttr_sc = [ 0.5, 1.5 ],
mttf_nc = [ 394.2 , 1182.6 ],
mttr_nc = [ 0.5, 1.5 ],
mttf_clc = [ 394.2 , 1182.6 ] ,
mttr_clc = [ 0.5, 1.5 ],
mttf_cc = [ 394.2 , 1182.6] ,
mttr_cc = [ 0.5, 1.5 ],
mttf_walrus = [ 394.2 , 1182.6 ],
mttr_walrus = [ 0.5, 1.5 ]
)
```

Listing E.2: Script Mercury para análise de sensibilidade da arquitetura A2

```
percentageDifference (
model_ = "A2",
metric_ = "av",
```

```
samplingPoints = 10,
parameters = (
mttf_node1 = [ 88.61 , 265.852 ],
mttr_node1 = [ 0.17515 , 0.52545 ],
mttf_node2 = [ 88.61, 265.852 ],
mttr_node2 = [ 0.17515 , 0.52545 ] ,
mttf_frontend = [ 90.36 , 271.08 ] ,
mttr_frontend = [ 0.48435, 1.45305 ]
)
```

Listing E.3: Script Mercury para análise de sensibilidade do modelo de disponibilidade orientada a capacidade

```
percentageDifference (
model_ = "Model",
metric_ = "COA",
samplingPoints = 10,
parameters = (
MTTF_VM = [ 1440, 4320 ],
MTTR_VM = [ 0.0095, 0.0285 ],
MTTF_NODE = [ 242.405, 727.215 ],
MTTR_NODE = [ 0.455, 1.365 ]
),
output = (
type = "R",
yLabel = "COA",
baselineValue = COA,
format = "png"
)
```

F

Scripts para validação do SyncML

Listing F.1: Script para monitoramento do servidor de sincronização

```
#!/bin/bash
# Escreve o cabecalho de identificacao dos dados
echo "Start date time" >> monitoramento-service.txt

echo "Start date time"

while [ True ]
do

C=$((C+1))
FUNAMBOLSTATUS=$(nmap -p 8080 192.168.10.254 | grep 80/tcp | awk '{print $2}')

if [ "$FUNAMBOLSTATUS" == "closed" ] ; then
echo $C "DOWN" "Funambol Down" $(date | awk '{ print $2, $3, $4 }') >> monitoramento-service.
    txt
fi
else
if [ "$FUNAMBOLSTATUS" == "open" ] ; then
echo $C "UP" "Up Funambol" $(date | awk '{ print $2, $3, $4 }') >> monitoramento-service.txt
else
echo $C "DOWN" "Down Funambol" $(date | awk '{ print $2, $3, $4 }') >> monitoramento-service.
    txt
fi
fi

sleep 5
done
```

Listing F.2: Script para injeção de falhas no servidor de sincronização

```
echo "Fault injection SyncML...."
echo "# Event GT Date" >> SyncML.log

C1=0

while [ true ] ;
do
C1=$((C1+1))
STATUSS=$(nmap -p 8080 192.168.10.254 | grep 80/tcp | awk '{print $2}')
```

```
SSTATUS=$(nmap -p 8080 192.168.10.254 | grep "Nmap done" | awk '{print $6}')

if [ "$SSTATUS" == "(1" ] ; then

if [ "$STATUSS" == "open" ] ; then
TIME=$(Rscript expFailSyncML.r | awk '{print $2}')
sleep $TIME
./SyncMLFailure.sh
echo "ok-F" >> SyncML.log
echo "Fault injected"
echo  $C1 "Fault" $TIME $(date | awk '{ print $2, $3, $4 }') >> SyncML.log

fi

if [ "$STATUSS" == "closed" ] ; then
TIMER=$(Rscript expRepairSyncML.r | awk '{print $2}')
sleep $TIMER
./SyncMLRepair.sh
echo "ok-R" >> SyncML.log
echo "Repair Injected"
echo $C1 "Repair" $TIMER $(date | awk '{ print $2, $3, $4 }')  >> SyncML.log
fi
done
echo "Fim"
```