Pós-Graduação em Ciência da Computação

Igor de Oliveira Costa

MODELOS PARA ANÁLISE DE DISPONIBILIDADE EM UMA PLATAFORMA DE *MOBILE BACKEND AS A SERVICE*

Dissertação de Mestrado

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

RECIFE
2015

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

Igor de Oliveira Costa

MODELOS PARA ANÁLISE DE DISPONIBILIDADE EM UMA PLATAFORMA DE *MOBILE BACKEND AS A SERVICE*

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

Orientador: *Prof. Dr. Paulo Romero Martins Maciel*

RECIFE
2015

Igor de Oliveira Costa

Modelos para análise de disponibilidade em uma plataforma de *Mobile Backend as a Service*/ Igor de Oliveira Costa. – RECIFE, 2015-

117 p. : il. (algumas color.) ; 30 cm.

Orientador Prof. Dr. Paulo Romero Martins Maciel

Dissertação de Mestrado – Universidade Federal de Pernambuco, 2015.

1. Palavra-chave1. Modelos Analíticos 2. Palavra-chave2. *Mobile Backend-as-a-Service* 3. Palavra-chave3. *Mobile Cloud Computing* I. Orientador. Paulo Romero Martins Maciel II. Universidade Federal de Pernambuco. III. Título Modelos para análise de disponibilidade em uma plataforma de Mobile Backend as a Service

CDU 02:141:005.7

Dissertação de mestrado apresentada por **Igor de Oliveira Costa** ao programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **Modelos para análise de disponibilidade em uma plataforma de *Mobile Backend as a Service***, orientada pelo **Prof. Dr. Paulo Romero Martins Maciel** e aprovada pela banca examinadora formada pelos professores:

Prof. Djamel Fawzi Hadj Sadok
Centro de Informática/UFPE

Prof. Almir Pereira Guimarães
Instituto de Computação/UFAL

Prof. Paulo Romero Martins Maciel
Centro de Informática - UFPE

RECIFE
2015

Dedico esta dissertação a minha esposa e aos meus filhos.

Agradecimentos

Agradeço primeiramente ao Deus todo poderoso que me permitiu chegar até aqui e cumprir mais uma etapa da minha vida, pois nele encontrei forças para continuar e não desistir.

A minha família, que sem dúvida é meu alicerce, minha fortaleza, onde encontro aconchego e paz, em especial a minha amada esposa Mariana Silva de Lima, e minhas filhas Maria Helena e Maria Celine que daqui 4 meses virá ao mundo alegrar nossos dias. Em fim, a todos os meus familiares.

A todos os meus amigos, que de forma direta ou indireta tiveram participação importante nessa caminhada, em especial aos meus amigos de MoDCS e mestrado, Eliomar, Verônica, Rosângela, Júlio, Carlos, Clara, Jamilson, Jean, Airton, Jonathan e Leandro que contribuíram de forma direta para essa conquista, agradeço de coração a vocês.

Agradeço também a Faculdade Sete de Setembro (FASETE), que de forma direta me permitiu iniciar esse sonho.

Ao meu orientador Paulo Maciel, pela disponibilidade, compreensão e paciência, e pelos valiosos ensinamentos, que com certeza levarei para minha vida enquanto professor. Gostaria de agradecer ao Conselho Nacional de Desenvolvimento Científico e Tecnológico CNPq pelo suporte financeiro prestado durante esse período.

Os analfabetos do próximo século não são aqueles que não sabem ler ou escrever, mas aqueles que se recusam a aprender, reaprender e voltar a aprender.

—ALVIN TOFFLER

Resumo

As limitações da computação móvel abrem caminhos para utilizar recursos de computação em nuvem voltadas à dispositivos móveis, sendo este o principal objetivo da *Mobile Cloud Computing* (MCC). Questões como armazenamento e processamento podem afetar a disponibilidade de um serviço no dispositivo móvel, assim, para minimizar esses problemas é possível o particionamento da aplicação em *frontend* e *backend*. Os serviços de nuvem auxiliam esse processo com a utilização de ambientes *Mobile Backend-as-a-Service* (MBaaS), que permitem os desenvolvedores conectar o *backend* de suas aplicações para o armazenamento em nuvem. Uma plataforma de MBaaS oferece um serviço de sincronização completa para aplicações móveis. Uma vez que os dados armazenados no dispositivo móvel estão sincronizados com os centros de dados distribuídos, a disponibilidade do sistema no lado servidor é um atributo fundamental que requer investigação, pois sistemas computacionais tendem a falhar. As falhas podem ocorrer em *hardwares*, *softwares* e meios de conexão, acarretando assim, em prejuízos financeiros e comprometendo a credibilidade das empresas provedoras do serviço. Os administradores necessitam de mecanismos para estimar a disponibilidade de seus sistemas, podendo definir *Service Level Agreement* (SLA) com mais propriedade. Assim, modelos analíticos podem ser utilizados para avaliar a disponibilidade destes tipos de ambientes, bem como auxiliar a mitigar o *downtime*, aumentando a disponibilidade do serviço. Este trabalho propõe modelos analíticos para avaliar a disponibilidade desses ambientes. Para tanto, foi adotada uma metodologia: primeiramente definiu-se a arquitetura básica do serviço; a qual foi modelada a partir de um modelo hierárquico, composto de diagramas de blocos de confiabilidade (RBD) e cadeias de Markov de tempo contínuo (CTMC) e validado através de um *testbed* de injeção de falhas e reparos em um ambiente real. Baseado no modelo de serviço proposto, foi efetuada a análise de sensibilidade, que identificou o sistema como componente crítico. A partir disto, foram sugeridos modelos hierárquicos que representem o ambiente de nuvem, e com base neste ambiente, através da técnica de análise de sensibilidade, foram propostas quatro arquiteturas, sendo estas avaliados em termos de disponibilidade e *downtime* anual. Os resultados demonstram que a implementação de um processo de recuperação automática sobre o componente de *software*, *Java Virtual Machine*, reduz o *downtime* anual na arquitetura básica em cerca de 10%, bem como é possível observar que no ambiente de nuvem utilizando o mecanismo de redundância *warm-standby* nos nós e no *frontend* apresenta efetiva melhora na disponibilidade. Desta forma, a presente pesquisa pode orientar os administradores de sistemas MBaaS no planejamento de suas políticas de manutenção.

Palavras-chave: *Mobile Backend-as-a-Service*. Dispositivos Móveis. Modelos Analíticos. *Mobile Cloud Computing*. Análise de Disponibilidade

Abstract

The mobile computer restrictions propose new ways to use cloud computing resources aimed at mobile devices, this is the Mobile Cloud Computing (MCC) primary goal. Issues such as storage and processing can impact the service availability on the mobile device. With the reducing purpose, these questions are its possible divide the application into two pieces, frontend, and backend. The cloud services assist this process with the Mobile Backend-as-a-Service (MBaaS) use. This tool allows the developers connect yours application backend to cloud storage. The MBaaS OpenMobster platform offers complete synchronization service to mobile applications. Since the data stored on mobile was synchronized distributed data center, the server's system availability is an essential attribute and request attention, because computer systems will sometimes fail. The failures can happen on components variety as hardware, software and connections, causing financial losses and reliability compromising of the companies, which offer this services. The administrators need tools to project the system availability, with this they can define the SLA with more assurance. Analytic models can be used to availability evaluate in this environment and mitigate the downtime risk, this improves the service availability. This work primary goal is proposed analytic models to availability evaluated in these environments. It was adopted a methodology as follow: First, define the base service architecture. It was modeled by use a hierarchical model, using a reliability block diagram (RBD) and continuous-time Markov chain (CTMC). The validation considers a fault injection testbed and repairs on real environment. Considering the model proposed, it was done sensitivity analysis, these results present the system as a critical component. This analysis was proposed hierarchical models to represents cloud environment. On these sensibility analysis, a background was offered four scenarios. The scenarios were evaluated to determine the availability and annual downtime. The results show that the an automatic recovery implementation process on the software component, Java Virtual Machine, decrease the annual downtime on base architecture to 10%. The results present the availability improvement when adopted redundancy strategy as warm standby on a cloud environment. This way, the work can guide the MBaaS system administrators in planning their maintenance policies.

Keywords: MBaaS. Mobile Devices. Analytic Models. Mobile Cloud Computing. Availability Analysis

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

SSH *Secure Shell* 79

TI Tecnologia da Informação 28

VM *Virtual Machine* 27

Wi-Fi *Wireless Fidelity* 21

SRN *Stochastic Reward Net* 20

Sumário

1 Introdução

A computação móvel (PANDEY; NEPAL, 2012) vem ganhando cada vez mais espaço entre as pessoas, o que faz aumentar a demanda sobre aplicações móveis, tornando este um mercado emergente. A estimativa é que até 2017 os usuários de *smartphones* passem de 69,4% da população mundial (BOYLE, 2014). O mercado de aplicações continua em constante evolução, sendo estimado que o mercado global de aplicativos cresça de US$ 6,8 bilhões em 2010 para US$ 25,0 bilhões em 2015 (MARKETSANDMARKETS, 2010). Paralelamente ao crescimento da computação móvel, a computação em nuvem (SAREEN, 2013) vem ganhando destaque e popularidade nos últimos anos. Neste contexto, surge o paradigma da *Mobile Cloud Computing* (MCC) (HOANG et al., 2013), que tem como principal objetivo utilizar os recursos de computação em nuvem para superar atributos limitantes da computação móvel, como por exemplo, o poder de processamento, capacidade de memória e armazenamento, permitindo a entrega de aplicações mais sofisticadas e inovadoras para o usuário.

O rápido progresso da computação móvel também tange o âmbito empresarial, que com o avanço da tecnologia móvel tende a transferir suas bases de dados para centros de dados distribuídos, provendo assim serviços através de aplicações móveis. Neste contexto, um novo serviço conhecido como *Mobile Backend-as-a-Service* (MBaaS) está emergindo (ROWINSKI, 2012), este por sua vez faz parte do âmbito da MCC. MBaaS, trata-se de um modelo que permite aos desenvolvedores vincular o *Backend* de suas aplicações ao armazenamento em nuvem. Fornece recursos de gerenciamento de usuários e dispositivos que estarão aptos a utilizar a aplicação, notificações por *push* e integração com serviços em redes sociais (SAREEN, 2013).

Esta tecnologia forma uma espécie de ponte entre a interface de uma aplicação e o *backend* baseado na nuvem, e isso é feito por meio de uma *Application Programming Interface* (API) unificada (LANE, 2012). Um dos objetivos da MBaaS é prover aos desenvolvedores de aplicações móveis um conjunto de funções para conectar suas aplicações a serviços disponíveis na nuvem (ROWINSKI, 2012), desta maneira, fornece uma maneira consistente para gerenciar os dados presentes na nuvem. A MBaaS é uma categoria de serviço relativamente nova que tem como principal objetivo suportar os *Mobile Enterprise Application Platforms* (MEAPs) (HELLER, 2014). O mercado global de MBaaS teve um crescimento de US$ 216,5 milhões

de dólares em 2012 para US$ 7,7 bilhões de dólares em 2017, o que representa uma taxa de crescimento anual composta de 104% entre 2012 e 2017 (MARKETSANDMARKETS.COM, 2012). Neste cenário, os serviços providos através dessas plataformas devem ser suficientemente confiáveis, ou seja, eles precisam garantir a dependabilidade. A dependabilidade de um sistema computacional é a capacidade de oferecer um serviço confiável, este conceito engloba vários atributos, como confiabilidade, disponibilidade, segurança, confidencialidade, integridade e manutenabilidade (AVIZIENIS; LAPRIE; RANDELL, 2001).

Com a utilização de um serviço baseado em MBaaS, o aplicativo estará dividido em *frontend*, que ficará no dispositivo, isto é, a parte gráfica da aplicação e o *backend*, que será movido, fazendo com que todo o processamento e armazenamento desse aplicativo aconteça na nuvem. Dentre os atributos da dependabilidade, a disponibilidade passa a ser importante item para observação, definindo-se como o fato de um serviço executar a função ao qual foi projetado em qualquer instante de tempo. De acordo com (TEOREY; NG, 1998), a disponibilidade pode ser medida, em geral, pelo número total de êxitos para o número total de tentativas; em outras palavras, é a razão do trabalho efetivo para o serviço pretendido, a fim de garantir a confiança dos usuários na aplicação.

Considerando a arquitetura de um serviço MBaaS, garantir sua disponibilidade também está relacionado a verificar os processos do lado servidor e prover uma infraestrutura que possibilite níveis aceitáveis de disponibilidade. Muitos são os problemas que podem afetar um provedor de serviço MBaaS, desde falhas no *hardware* e sistema operacional até falhas na aplicação e banco de dados. De acordo com (DANTAS et al., 2012) sistemas computacionais, bem como seus componentes de *software*, *hardware* e meios de interconexões (cabos e meios sem fio), tendem a falhar inesperadamente, tornando o sistema indisponível por algum tempo.

O trabalho ora posto concentra esforços na proposição de modelos analíticos com o intuito de investigar o comportamento da disponibilidade de uma categoria de serviço específica, a MBaaS. A pesquisa desenvolvida aborda o lado servidor do serviço, para tal, utiliza a plataforma *open source* de MBaaS OpenMobster, uma plataforma voltada para o âmbito empresarial. Seu objetivo principal é atender as necessidades das empresas através da mobilização de suas bases de dados (OPENMOBSTER, 2014a). Utilizamos uma metodologia que inicialmente consistiu em determinar uma arquitetura com requisitos mínimos para execução do serviço, ou seja, uma arquitetura básica da plataforma de MBaaS, desta forma, uma abordagem de modelagem hierárquica heterogênea utilizando modelos *Reliability Block Diagrams* (RBD) e *Continuous-Time Markov Chain* (CTMC) foi empregada com o intuito de investigar a disponibilidade do ambiente, onde o RBD foi usado para gerar o modelo de disponibilidade da infraestrutura, e a CTMC para gerar o modelo de disponibilidade do serviço MBaaS. Para a validação do modelo da arquitetura básica foi construído um experimento utilizando técnicas de injeção de falhas, como também foi utilizado o método proposto (KEESEE, 1965), para obter o intervalo de confiança da disponibilidade. Com o modelo validado e com o intuito de alcançar maior disponibilidade no serviço, propomos a utilização de um processo de recuperação automática na *Java Virtual Ma-*

chine (JVM). Ainda na arquitetura básica, por meio de uma análise de sensibilidade paramétrica foi avaliado o impacto dos diferentes parâmetros sobre a disponibilidade dos componentes do sistema em geral.

Através do modelo de serviço validado e da análise de sensibilidade efetuada, propomos a implementação do serviço em um ambiente de nuvem privada *Eucalyptus*, com o objetivo de investigar melhorias que promovam um aumento da disponibilidade do sistema. Então, definimos uma arquitetura de nuvem sem redundância, a partir desta arquitetura foi realizada uma análise de sensibilidade para verificar gargalos de disponibilidade, e assim geramos mais três arquiteturas: A primeira uma arquitetura com redundância no *frontend* da nuvem (DANTAS, 2013); A segunda uma arquitetura com redundância nos nós, utilizando um mecanismo de redundância *warm standby*, e a Terceira arquitetura abordou a redundância nos nós e no *frontend* da nuvem.

1.1 Motivação e Justificativa

Atualmente a adoção de aplicações empresariais móveis tem se tornado uma necessidade estratégica nas organizações, pois o crescente avanço da computação móvel conscientizou as organizações a implementar aplicativos móveis para funcionários internos, fornecedores e clientes com o objetivo de mobilizar seus processos (BAL, 2013). No entanto, as limitações de capacidade dos dispositivos móveis como, por exemplo, armazenamento, processamento, e bateria limitam as funcionalidades dessas aplicações. Neste contexto, a MCC e seus serviços tais como MBaaS passam a ser importantes aliados a computação móvel, pois além de oferecer uma infraestrutura para armazenamento em nuvem, ainda possibilitam o *offloading* das aplicações, que consiste em particionar uma tarefa em componentes que podem executar tanto no dispositivo móvel quanto na nuvem (KUMAR; LU, 2010), ou também mover o *backend* das aplicações para a nuvem.

Observando esse cenário, podemos verificar que os atributos de dependabilidade como disponibilidade, confiabilidade e segurança podem ser apontados como atributos críticos em ambientes empresariais que utilizam uma infraestrutura baseada em MCC, e dependem da disponibilidade de seus serviços para a execução de seus processos administrativos. Assim, um sistema baseado em MCC que venha a falhar, ou seja, que não cumpra níveis satisfatórios de disponibilidade, pode gerar prejuízos e comprometer a credibilidade dos usuários na aplicação (OLIVEIRA et al., 2013).

Projetar uma infraestrutura que seja confiável passa a ser essencial para que as organizações evitem ou minimizem possíveis prejuízos. Construir um ambiente de alta disponibilidade requer aplicar redundância em componentes que compõem esse ambiente, porém o uso errado de redundância de componentes pode aumentar o custo com aquisição de recursos computacionais, gerando possíveis prejuízos (DANTAS et al., 2012). A modelagem analítica é um meio que possibilita a avaliação de diversos tipos de sistemas (KIM; MACHIDA; TRIVEDI, 2009a), inclu-

sive os que exijam alta disponibilidade, pois utiliza simulação para avaliar possíveis gargalos no sistema, e desta maneira, poder empregar redundância no componente correto, minimizando o risco de erros no projeto.

Portanto, torna-se importante a elaboração de modelos analíticos que tenham o objetivo de auxiliar os administradores de infraestruturas de MBaaS, a avaliar e projetar um ambiente que cumpra níveis satisfatórios de disponibilidade, minimizando assim possíveis prejuízos nas organizações que adotem esta categoria de serviço.

1.2 Objetivos

O principal objetivo deste trabalho é a proposição de um conjunto de modelos hierárquicos baseados em CTMC e RBD para a avaliação de disponibilidade de infraestruturas MBaaS. A finalidade é auxiliar o planejamento das infraestruturas MBaaS, bem como na definição de políticas de manutenção.

A fim de alcançar objetivo geral, os seguintes objetivos específicos são detalhados:

a) Propor uma metodologia de avaliação de disponibilidade para ambientes MBaaS;

b) Construir um *testbed* de injeção e monitoramento de falhas para ambientes MBaaS;

c) Validar o modelo da arquitetura básica, através do experimento de injeção de falhas;

d) Encontrar gargalos de disponibilidade na arquitetura básica através do uso de análise de sensibilidade paramétrica;

e) Propor alternativas de infraestruturas à arquitetura básica do serviço MBaaS.

1.3 Trabalhos Relacionados

Avaliação de disponibilidade em ambientes MCC tem sido tema de pesquisas que abordam a área em diversas perspectivas, no entanto, foram encontrados poucos trabalhos que abordem a utilização de modelagem analítica e técnicas de injeção de falhas nesses ambientes. Em pesquisas realizadas nas *engines* IEEEXplore, ACM, Scopus, ScienceDirect, SpringerLink e em trabalhos publicados no período de 2011 a 2015, não foram encontrados, até o momento, trabalhos que tratem a análise de disponibilidade em plataformas de MBaaS. Desta forma, classificamos os trabalhos relacionados em: avaliação de dependabilidade em *Cloud Computing*, infraestrutura de MCC e avaliação de disponibilidade em MCC, apresentados nas seguintes seções.

1.3.1 Avaliação de Dependabilidade em *Cloud Computing*

O trabalho desenvolvido por (MELO et al., 2014) apresenta um modelo de disponibilidade de um serviço de *streaming* de vídeo implantado em um ambiente de nuvem privada. A estratégia de modelagem utilizada combina de forma hierárquica os formalismos RBD e CTMC, onde os componentes principais da nuvem foram representados por RBD, enquanto o serviço de *streaming* de vídeo foi refinado com uso de CTMC devido a sua complexidade. A fim de orientar a melhoria da disponibilidade do sistema, foi aplicada uma análise de sensibilidade diferencial, com o objetivo de identificar os componentes críticos do sistema do ponto de vista da disponibilidade, dessa forma mecanismos de redundância na infraestrutura foram propostos. Os resultados demonstraram que uma estratégia de modelagem combinada com análise de sensibilidade diferencial pode ser uma metodologia atrativa para identificar os componentes que aplicando redundância possam aumentar a disponibilidade do sistema.

Os benefícios de um mecanismo de replicação *warm-standy* em um ambiente de computação em nuvem privada Eucalyptus foi investigado por (DANTAS, 2013). Os autores utilizaram uma abordagem de modelagem hierárquica heterogênea para representar a arquitetura redundante e comparar a disponibilidade desse ambiente com de ambientes sem redundância. As falhas de *hardware* e *software* são consideradas nos modelos analíticos propostos, os quais também são usados para obter equações de fórmulas fechadas a fim de calcular a disponibilidade de infraestrutura de nuvem. Os resultados desta pesquisa demostram que a adoção de um ambiente de nuvem redundante pode acarretar em melhorias consideráveis para disponibilidade do ambiente.

Uma das principais características de uma infraestrutura de nuvem é a alta disponibilidade. (LONGO et al., 2011) apresenta um método escalável para análise de disponibilidade em uma nuvem *Infrastructure as a Service* (IaaS). Os autores utilizam modelos analíticos baseados em *Stochastic Petri Net* (SPN). Uma das principais contribuições deste trabalho está na modelagem do sistema que apresenta um conjunto de macro-modelos baseados em *Stochastic Reward Net* (SRN) interagindo com submodelos baseados em CTMC. Desta forma, gera modelos escaláveis para avaliação de disponibilidade em ambientes de computação em nuvem. Através dos submodelos baseados em CTMC foi desenvolvido soluções de fórmulas fechadas, bem como uma análise de sensibilidade aplicada em alguns parâmetros. Neste trabalho nenhum método de validação dos modelos propostos foi abordado.

Modelos de disponibilidade para ambiente de computação em nuvem baseado em RBD e Redes de Petri são apresentados por (MELO et al., 2013) e foram concebidos com o objetivo de avaliar o mecanismo de *live migration* com rejuvenescimento baseado em tempo. A principal contribuição deste trabalho é avaliar o impacto que diferentes políticas de rejuvenescimento, com base no mecanismos de *live migration*, produzem sobre a disponibilidade. Através de uma análise de sensibilidade, é observado que a disponibilidade de estado estacionário é substancialmente melhorada com um intervalo de tempo apropriado para as ações de rejuvenescimento.

Os trabalhos destacados nesta subseção têm como principal objetivo a proposta de

modelos voltados à obtenção de alta disponibilidade em ambiente de nuvens, e não apresentam uma abordagem para validação destes modelos e a utilização de técnicas de injeção de falhas, a fim de verificar o comportamento do sistema. Tais trabalhos apresentam um conjunto de fórmulas fechadas que podem ser úteis para representar a disponibilidade do sistema. Assim esta pesquisa apresenta um conjunto de modelos de disponibilidade para ambiente de nuvem, focado em uma categoria de serviço específica, o MBaaS. No entanto, é utilizado uma abordagem de injeção de falhas para a validação do modelo de serviço, além da investigação sobre o benefício da implementação de um processo de recuperação automática em um dos componentes do serviço.

1.3.2 Infraestrutura de MCC e Avaliação de Disponibilidade em MCC

SANAEI et al. (2012) propõe um modelo arbitrado de infraestrutura multicamadas baseado em *Service Oriented Architecture* (SOA), denominado SAMI. Neste modelo são definidas três camadas principais: SOA, *arbitrator*, e *infrastructure*. Sua principal contribuição está na camada *infrastructure*, uma camada *multi-tier* que aproveita infraestruturas de três provedores de serviço de nuvem e provedores de redes móveis. Logo acima da camada *infrastructure* existe a camada *arbitrator* que funciona como uma espécie de *gateway*, na qual classificam-se as solicitações de serviços e lhes destina os recursos adequados, com base em vários indicadores, tais como: exigência de recursos, latência e segurança. Este trabalho visa estabelecer uma interoperabilidade de baixo acoplamento entre clientes móveis, provedores de nuvem e provedores de redes móveis. No entanto, por se tratar de uma arquitetura complexa de três camadas, um estudo sobre a disponibilidade traria um contribuição relevante para pesquisa, dando melhor apoio sobre o benefício dessa arquitetura.

A utilização de modelagem hierárquica para um ambiente de *mobile cloud* é abordado no trabalho de (ARAUJO et al., 2014). Nele, os autores propõem um modelo de alto nível que caracteriza o comportamento de um sistema *mobile Health* (mHealth) baseado em uma infraestrutura de *mobile cloud*. Para tanto, foi necessário a proposição de modelos que representem todo o domínio de uma *mobile cloud*, incluindo: infraestrutura de nuvem, comunicação sem fio, e dispositivos móveis. O modelo de nuvem foi dividido em vários submodelos RBD. Foram considerados os principais componentes de uma infraestrutura de nuvem, tais como: gerenciador de infraestrutura, gerenciador de armazenamento e gerenciador do nó. Os modelos de comunicação foram concebidos utilizando o formalismo de Redes de Petri, representando a comunicação de dois mecanismos sem fio (*Wireless Fidelity* (Wi-Fi) e 4G). Os principais componentes do dispositivo móvel foram representados pelos autores utilizando RBD, considerando vários cenários, diferentes possibilidade de comunicação, taxas de descarga de bateria e diferentes tempos de espera. Os resultados mostraram que o tempo de espera é a métrica com maior impacto na disponibilidade do sistema mHealth e que as baterias mais eficientes ajudam a melhorar a disponibilidade do sistema. Todavia, este trabalho não aborda uma metodologia de validação dos modelos propostos.

Em (PANDEY; NEPAL, 2012), foram propostas métricas de disponibilidade definida em função de um perfil de disponibilidade, modelo de custo e utilização de serviço. O autor define um modelo baseado em CTMC para avaliar a disponibilidade dos serviços de dados através de múltiplos fornecedores, com o objetivo de tornar os serviços altamente disponíveis à um custo menor. Este trabalho analisa a disponibilidade do serviço, com base em perfis de disponibilidade definidos pelo usuário e sua relação com a receita esperada por um prestador de serviço de dados em nuvem. No entanto, os componentes do lado do servidor são abstraídos. Em nossa pesquisa, o foco principal são os componentes no lado servidor, pois como já foi explicado anteriormente, esses componentes tendem a falhar impactando assim na disponibilidade do serviço provido.

MATOS et al. (2014) utiliza modelagem hierárquica com RBD e CTMC e quatro técnicas de análise de sensibilidade diferentes, a fim de determinar os parâmetros que causam o maior impacto sobre a disponibilidade de uma nuvem móvel. Esta pesquisa aborda todo o domínio de uma MCC, tais como os clientes móveis, a comunicação, e infraestrutura de nuvem. Nos clientes móveis são considerados os componentes: *hardware*, sistema operacional, bateria, e aplicação. Na comunicação são considerados os componentes 3G e Wi-Fi, e na infraestrutura de nuvem são abordados os componentes *Infrastructure Manager*, *Storage Manager* e cinco nós. Os resultados obtidos demonstram que com específicas exceções, abordagens distintas fornecem resultados semelhantes quanto à classificação de sensibilidade, entretanto, o trabalho não foca em nenhuma categoria de serviço específica.

Em (OLIVEIRA et al., 2013) é realizado um estudo da disponibilidade para ambientes MCC em diferentes cenários, considerando as tecnologias de comunicação sem fio, tais como: 3G e Wi-Fi. Utiliza-se modelagem hierárquica, que combina redes de Petri e modelos RBD. Aqui, avaliou-se o impacto da comunicação sem fio e do consumo de energia sobre a disponibilidade de sistemas. Os resultados demonstram que o protocolo 3G provoca uma menor disponibilidade e também é o principal responsável pela descarga da bateria, porém, quando combinado com o protocolo Wi-Fi em uma redundância de comunicação, mostra melhores resultados de disponibilidade e de tempo de operação de bateria. Como no trabalho de (MATOS et al., 2014), este também não foca em nenhuma categoria de serviço específica.

As pesquisas apresentadas em (ARAUJO et al., 2014), (PANDEY; NEPAL, 2012), (MATOS et al., 2014) e (OLIVEIRA et al., 2013) têm como objetivo a proposição de modelos para a avaliação de disponibilidade em ambiente de MCC. Destas, apenas a de (PANDEY; NEPAL, 2012) abstrai o lado servidor, sem ênfase a este domínio. Os outros trabalhos abordam todo o domínio de uma MCC; no entanto, apenas (ARAUJO et al., 2014) aborda um serviço específico na modelagem do ambiente. Nenhum dos trabalhos citados propõe uma metodologia de validação dos modelos. Não obstante, e diferente dos trabalhos acima apresentados, esta pesquisa concentra esforços na categoria de serviço MBaaS, através da plataforma OpenMobster, dando enfoque aos componentes do lado servidor e propondo modelos hierárquicos e heterogêneos de disponibilidade em vários cenários, com o intuito de apoiar a avaliação de disponibilidade da plataforma de MBaaS em execução nesses cenários.

A Tabela 1.1 apresenta um comparativo entre os trabalhos relacionados e a pesquisa desenvolvida. O principal objetivo é destacar os pontos fortes e principais limitações da presente pesquisa em relação às demais. Como mencionado anteriormente, não foi encontrado trabalhos publicados que abordem o domínio da nossa pesquisa, a proposição de modelos analíticos para a avaliação de disponibilidade em um ambiente MBaaS. Isto pode ser justificado por se tratar de uma categoria de serviço relativamente nova. No entanto, podemos verificar algumas limitações da presente pesquisa em relação aos trabalhos apresentados. Estas limitações se dão por conta do propósito da pesquisa desenvolvida, que tem como objetivo principal o lado servidor do domínio de uma MCC, focando na categoria de serviço MBaaS. Desta forma, podemos afirmar que a presente pesquisa trata aspectos relevantes em abordar uma área ainda pouco explorada. Além disso, propõe modelos analíticos para ambientes MBaaS e apresenta um experimento de validação do modelo na arquitetura básica; acrescido de uma análise de sensibilidade no ambiente, a fim de verificar gargalos de disponibilidade.

Tabela 1.1: Comparação Trabalhos Relacionados

Trabalhos Relacionados	Contexto	Modelos Analíticos	Análise de dependabilidade	Análise de sensibilidade	Validação do modelo	Injeção de Falha	Modelagem Hierárquica	MBaaS
Esta pesquisa	MCC e *Cloud* Privada	✓	disponibilidade e indisponibilidade	✓	✓	✓	RBD, CTMC	✓
(MELO et al., 2014)	*Cloud* Privada	✓	disponibilidade e indisponibilidade	✓			RBD, CTMC	
(DANTAS, 2013)	*Cloud* Privada	✓	disponibilidade e indisponibilidade				RBD, CTMC	
(LONGO et al., 2011)	*Cloud*	✓	disponibilidade	✓			SPN, CTMC	
(MELO et al., 2013)	*Cloud* Privada	✓	disponibilidade e indisponibilidade	✓			SPN, RBD	
(SANAEI et al., 2012)	MCC							
(ARAUJO et al., 2014)	MCC	✓	disponibilidade				RBD, SPN	
(PANDEY; NEPAL, 2012)	MCC	✓	disp.					
(MATOS et al., 2014)	MCC	✓	disponibilidade e indisponibilidade	✓			RBD, CTMC	
(OLIVEIRA et al., 2013)	MCC	✓	disponibilidade e indisponibilidade		✓	✓	RBD, SPN	

1.4 Estrutura da Dissertação

A pesquisa está organizada da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica sobre os temas abordados neste trabalho: *Mobile Cloud Computing* (MCC), Plataforma de MBaaS, fazemos uma discussão sobre dependabilidade, em particular, sobre o conceito de disponibilidade, essencial neste trabalho; seguida pela abordagem sobre as técnicas de modelagem utilizadas para realizar a avaliação de atributos de dependabilidade, como também apresentamos conceitos de análise de sensibilidade e técnicas de injeção de falhas. O Capítulo 3 destaca as arquiteturas MBaaS propostas, como também apresenta a metodologia aplicada. O Capítulo 4 detalha os modelos analíticos propostos com base nas arquiteturas apresentadas. O Capítulo 5, expõe o processo de validação do modelo da arquitetura básica. O Capítulo 6 apresenta os estudos de caso desta pesquisa, baseados nos modelos propostos. Por fim, o Capítulo 7 apresenta as conclusões, bem como os trabalhos futuros a serem desenvolvidos com base nessa pesquisa.

2

Fundamentação Teórica

Este capítulo apresenta os conceitos básicos sobre os temas abordados nesta dissertação relevantes para a compreensão deste trabalho. Inicialmente, abordamos os conceitos relacionados a *Mobile Cloud Computing* e *Cloud Computing* e logo após apresentamos a plataforma de MBaaS OpenMobster, utilizada nessa pesquisa. Apresentamos também os conceitos básicos de dependabilidade e as técnicas de avaliação utilizadas neste trabalho. Por último, e não menos importantes, apresentamos uma explicação básica sobre análise de sensibilidade e conceituamos técnica de injeção de falhas.

2.1 *Cloud Computing*

Mobile Cloud Computing é a integração da computação móvel com a computação em nuvem, sendo assim, esta seção traz os principais conceitos relacionados a *Cloud Computing* e logo após faz uma abordagem sobre MCC.

A computação em nuvem é um modelo utilizado para habilitar o acesso ubíquo, conveniente e sob demanda a um conjunto de recursos computacionais compartilhados (rede, servidores, armazenamento, aplicações e serviços), que pode ser rapidamente provido e liberado com o mínimo de esforço gerencial (MELL; GRANCE, 2011). Segundo (ARMBRUST et al., 2010) computação em nuvem é um modelo computacional no qual recursos como, poder de processamento, rede, armazenamento e softwares são oferecidos através da Internet e podem ser acessados remotamente. Dessa forma, este modelo permite que usuários obtenham os recursos de forma elástica, sob demanda e a um baixo custo, entregues de maneira semelhante a serviços tradicionais como água, gás, eletricidade e telefonia (DINH et al., 2013). Existem várias definições na literatura para o termo *cloud computing*. A definição de (VAQUERO et al., 2008) é uma destas; o autor afirma que *Clouds* são um grande conjunto de recursos virtualizados facilmente usáveis e acessíveis (tais como *hardware*, plataforma de implantação e/ou serviços). Estes serviços podem ser dinamicamente reconfigurados para uma carga variável, permitindo uma utilização ótima dos recursos.

Um dos principais vetores da computação em nuvem é a virtualização, assim como *grid*

computing, computação de alto desempenho, entre outros (GONG et al., 2010). Os recursos computacionais de uma *grid* são provisionados para os clientes como *Virtual Machine* (VM)s através da virtualização de uma infraestrutura de *data center*, de forma que cada cliente só pague pela quantidade de recursos que consumir, ao invés de pagar uma taxa fixa. Um grande número de corporações tendem a aderir a este modelo computacional, pois este possui características que o torna atrativo. Ao invés de manter uma infraestrutura local com todos os seus custos como equipamentos, energia e mão de obra, a empresa pode apenas contratar esse serviço através de um provedor de nuvem. Por exemplo, se uma empresa deseja lançar um serviço na Internet, pode simplesmente alugar um conjunto de VMs em um serviço na nuvem e hospedar seu serviço, sem a necessidade de manter sua própria infraestrutura de servidores (ZHANG; CHENG; BOUTABA, 2010). Isso ajuda a reduzir o *"time to market"* de seus produtos e consequentemente, os torna mais competitivos (OLIVEIRA et al., 2013). Em (VAQUERO et al., 2008) é definida algumas características essenciais para nuvens, são estas:

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

A evolução da tecnologia, o desenvolvimento de computadores cada vez mais potentes e a preocupação com os custos, fizeram com que a virtualização torne-se uma realidade presente nos *data centers*. A virtualização de recursos pode ser considerada uma metodologia que possibilita a divisão dos recursos de um computador em diversos ambientes de execução (APPARAO et al., 2008). Virtualização de recursos é uma tendência real nos *data centers*, pois permite a emulação de plataformas e ambientes que necessitam de maior desempenho computacional, além de acarretar benefícios como a consolidação de carga de diversos serviços subutilizados em poucos servidores, facilidade de gerenciamento, redução de custo de infraestrutura e consumo de energia.

A arquitetura baseada em serviço permite que clientes utilizem os serviços alocados em nuvem através de um *web browser*. Unindo a computação em nuvem e a arquitetura baseada em serviços. Aplicativos e demais recursos de TI podem ser oferecidos remotamente, como se estivessem localizados no ambiente local (MELO et al., 2013).

Uma das principais características do modelo de *cloud computing* é a elasticidade. (GONG et al., 2010) cita que uma das propostas da computação em nuvem é a impressão de recursos infinitos; tais recursos são disponibilizados sob demanda. Podemos definir a elasticidade como a capacidade do ambiente computacional da nuvem aumentar ou diminuir de forma automática os recursos computacionais demandados e provisionados para cada usuário.

Assim como diversos recursos, tais como: energia, telefone e água; o modelo de pagamento dos recursos de nuvem é baseado no consumo, ou seja, o usuário paga apenas pelo

que usar. Desta forma, o modelo de recursos em nuvem se torna atraente para os usuário de Tecnologia da Informação (TI), uma vez que permite a alocação de novos recursos somente quando necessário.

As nuvens computacionais podem ser classificadas sobre dois aspectos, um relativo ao ponto de vista do usuário e outro relativo ao modelo de negócio (ZHANG; CHENG; BOUTABA, 2010). No aspecto relativo ao ponto de vista do usuário as nuvens podem ser classificadas como:

- **Privada:** Infraestruturas de nuvem projetadas para serem usadas por uma única organização. Uma infraestrutura de nuvem privada pode ser provida por um agente externo ou ser gerenciada pela própria empresa. No último caso, essa opção é adotada por organizações que possuem elevado nível de criticidade dos dados como bancos, órgãos do governo, agências militares, dentre outros, não permitindo que estes dados sejam confiados a terceiros, no caso, provedores de nuvens públicas. A desvantagem na adoção desta abordagem em relação a nuvens públicas é a exigência do investimento inicial em um *server farm* proprietário (ZHANG; CHENG; BOUTABA, 2010), idêntico ao modelo tradicional sem uso de *cloud*.

- **Pública:** Nuvens mantidas por provedores de serviço, que disponibilizam seus recursos computacionais para outras organizações. Este tipo de nuvem reflete melhor os benefícios originais propostos pela computação em nuvem comparado às nuvens privadas. Diferente de uma nuvem privada, o investimento inicial de infraestrutura é zero, pois não é necessário adquirir servidores e outros equipamentos próprios. Nuvens públicas geralmente são oferecidas por grandes *data centers* que possuem capacidade computacional elevada ao de uma nuvem privada. A nuvem pública tem como vantagem delegar, para o provedor da nuvem, a responsabilidade pelo gerenciamento da infraestrutura e por manter o *Service Level Agreement* (SLA) combinado (CAROLAN; GAEDE, 2009). No entanto, empresas que detém dados sigilosos podem optar por um investimento em uma infraestrutura de nuvem privada, pois na nuvem pública não é possível obter um controle total sobre os dados, redes e atributos de segurança.

- **Híbrida:** É uma combinação dos modelos de nuvem pública e privada, a fim de superar as limitações de cada abordagem. A vantagem deste modelo é proporcionar o melhor dos dois mundos. Dados críticos de uma organização podem ser mantidos na parte privada da nuvem, ao mesmo tempo em que é possível escalar o serviço através da imensa capacidade de uma nuvem privada.

A Figura 2.1 detalha as nuvens no aspecto relativo ao ponto de vista do usuário.

No aspecto relacionado ao modelo de negócio, os serviços de computação em nuvem são amplamente divididos em três categorias:

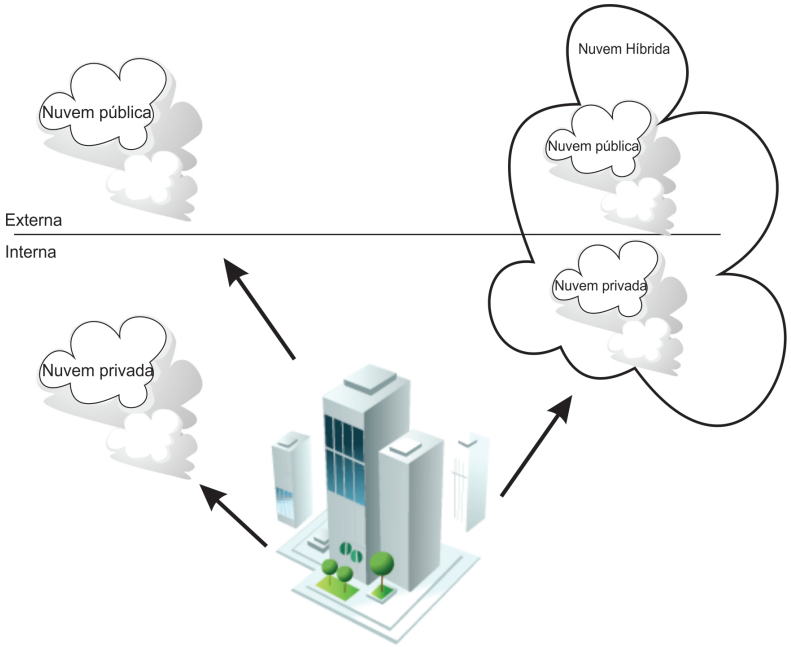


Figura 2.1: Tipos de nuvem adaptado de (FURTH; ESCALANTE, 2010)

- **IaaS**: Este modelo de negócio utiliza a virtualização dos recursos para particionar a infraestrutura de um *data center* entre os usuários, que por sua vez pagam apenas pelos recursos que utilizarem, ou seja, o usuário paga pelo serviço de acordo com a capacidade das VMs alocadas e pela quantidade de tempo que as VMs permanecem em execução.

- ***Platform as a Service* (PaaS)**: Neste modelo o provedor oferece uma plataforma de *software* de alto nível, abstraindo assim as VMs e seus componentes, como por exemplo o sistema operacional. De acordo com (VAQUERO et al., 2008) este modelo tem como vantagem o dimensionamento transparente dos recursos de hardware durante a execução dos serviços.

- ***Software as a Service* (SaaS)**: Com este modelo de negócio os usuários dispõem de um conjunto de *softwares* que podem ser acessados através da internet, a partir de um *web browser*. Assim, usuários da nuvem podem liberar seus aplicativos em um ambiente de hospedagem que pode ser acessado na rede por diversos clientes (DILLON; WU; CHANG, 2010).

As categorias relacionadas o modelo de negócios podem ser observadas no modelo de camadas descrito na Figura 2.2:

O modelos descritos são os principais e mais amplamente difundidos, no entanto, existem diversos outros modelos, como por exemplo, o *Desktop as a Service* (DaaS), *Data Storage as a Service* (DSaaS), e também o *Mobile Backend as a Service* (MBaaS), que faz parte do âmbito do paradigma de *Mobile Cloud Computing*. Basicamente, podemos definir como uma categoria de computação em nuvem que é composta por empresas que proporcionam facilidades aos

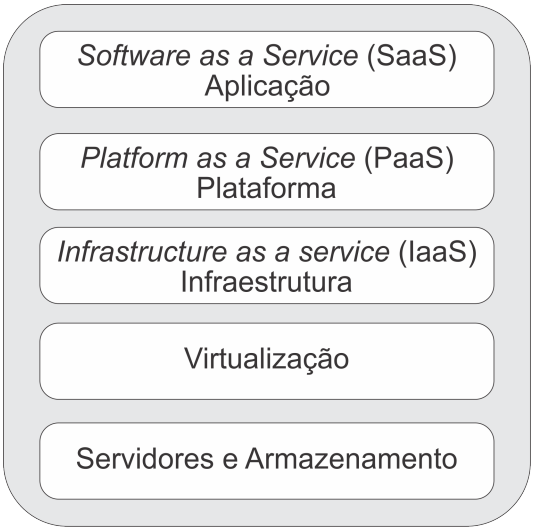


Figura 2.2: Camadas da arquitetura de *Cloud Computing*, adaptado de (FURTH; ESCALANTE, 2010)

desenvolvedores, tais como: instalação, utilização e operação de uma infraestrutura em nuvem para seus dispositivos móveis.

2.2 *Mobile Cloud Computing*

A computação móvel é cada vez mais essencial, como as ferramentas de comunicação mais eficazes e convenientes não limitadas por tempo e lugar (HOANG et al., 2013). Podemos definir a computação móvel a partir de duas propriedades principais. A primeira é que os dispositivos devem ser portáteis, de forma que suas dimensões reduzidas permitam à seus usuários permanecerem com tais dispositivos a maior parte do tempo. Esta propriedade é abreviada pela sigla *Bring Your Own Device* (BYOD) (ARCHER et al., 2012). A segunda propriedade, importante para a computação móvel, é a conectividade, isto é, a capacidade de receber e enviar dados para a Internet através de redes. Observando essas características a mobilidade é um requisito presente nestes dispositivos, que por sua vez, devido a características de tamanho, peso e outros, possuem limitações como: processamento, bateria e conexão. Contudo, as aplicações desenvolvidas para estes dispositivos precisam estar disponíveis para seus usuários o máximo de tempo possível, sendo assim, vários fatores podem afetar essa disponibilidade tanto do lado cliente, como por exemplo: conexão e bateria, como do lado servidor. Para tentar minimizar esses problemas e as limitações dos dispositivos, a computação móvel e a *Cloud Computing*, que ao contrário dos dispositivos móveis possuem um grande poder computacional e pode provê-lo sob demanda aos seus usuários, convergiram dando origem a uma nova área de pesquisa chamada *Mobile Cloud Computing* (QURESHI et al., 2011).

O principal objetivo da MCC é usar a capacidade de processamento e armazenamento de uma nuvem computacional para estender as capacidades de dispositivos móveis mais limitados, de forma a melhorar o desempenho e aliviar o consumo energético de aplicações móveis mais

pesadas. Este novo paradigma poderá permitir o provisionamento de aplicações móveis mais inteligentes, que estendam as capacidades cognitivas de usuários, tais como: reconhecimento de voz, processamento de linguagem natural, visão computacional, aprendizado de máquina, realidade aumentada, planejamento e tomada de decisão (SATYANARAYANAN et al., 2009).

A arquitetura básica de uma *Mobile Cloud Computing* é descrita na Figura 2.3. Onde é possível observar que a nuvem móvel é composta basicamente pelos mesmos componentes e categorias de serviços de uma *Cloud Computing* com acréscimo da categoria de serviço MBaaS, que faz parte do âmbito da *Mobile Cloud Computing* e tem como um de seus objetivos, facilitar a utilização da infraestrutura de nuvem através de uma API. Os dispositivos móveis se conectam à nuvem móvel através de uma infraestrutura de rede sem fio, podendo ser WLAN, 3G, 4G, GPRS e outras. De acordo com a arquitetura, uma vez que a computação é migrada para a nuvem, os requisitos sobre a capacidade dos dispositivos móveis são diminuídas, solucionando em partes o problema da limitação dos dispositivos móveis, onde uma gama maior de clientes pode ser atingida, como *smartphones* com maiores recursos, *smartphones* mais limitados, *tablets* e até mesmo alguns *feature phones* (QI; GANI, 2012). Com a nuvem sendo utilizada como plataforma de processamento das informações, tanto as redes sem fio, como a infraestrutura de nuvem passam a ser itens críticos na disponibilidade das aplicações. Dessa forma, prover uma infraestrutura que atenda a níveis aceitáveis de disponibilidade passa a ser um desafio para a *Mobile Cloud Computing*.

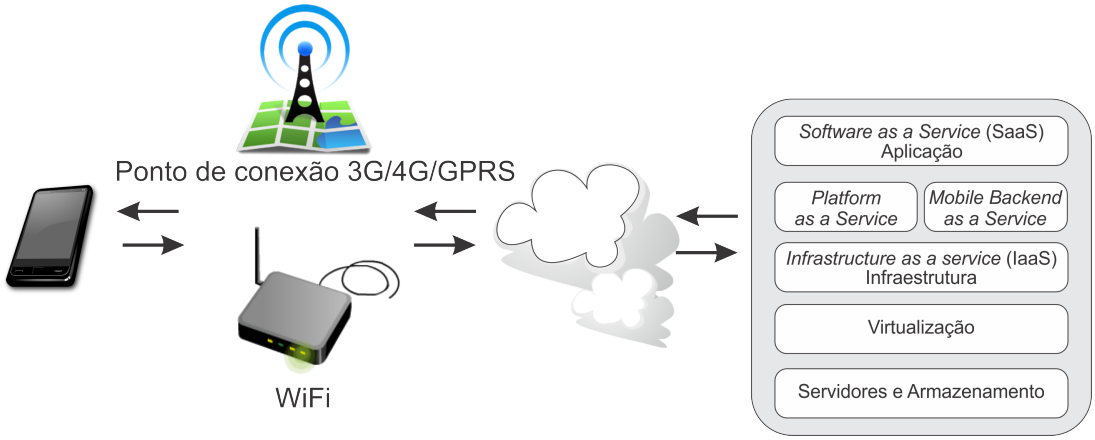


Figura 2.3: Arquitetura de *Mobile Cloud Computing*, adaptado de (QI; GANI, 2012)

2.3 Plataforma de MBaaS OpenMobster

A computação em nuvem não é apenas relacionada aos computadores pessoais, também afeta fortemente a tecnologia móvel (HOANG et al., 2013). Neste contexto, podemos destacar na MCC a categoria de serviço MBaaS, que é um modelo que através de API e *Software Development Kits* (SDKs) provê aos desenvolvedores de aplicativos móveis uma maneira de vincular o *backend* de suas aplicações ao armazenamento em nuvem. Ou simplesmente podemos

definir como uma arquitetura de computação que conecta aplicações móveis para serviços de computação em nuvem (ROUSE, 2015). Embora semelhante a outras categorias de serviços de computação em nuvem, como SaaS, IaaS, e PaaS, MBaaS é distinta dessas outras categorias de serviços, pois aborda especificamente a integração entre o *backend* de aplicações móveis e a *cloud computing*. Como já citado uma plataforma de MBaaS provê funcionalidade básica para aplicações móveis em nuvem, como por exemplo, gerenciamento de usuários e notificações *push*.

Existem diversas plataforma que oferecem o serviço baseado em *Mobile Backend as a Service*, como por exemplo, AnyPresence, Kinvey, Kumulos, Microsoft Azure e FeedHenry, são algumas, contudo essas citadas são soluções proprietárias, e sendo assim não fazem parte do escopo definido para a presente pesquisa, que por sua vez procurou focar em soluções *open source*, dentre essas podemos citar algumas como Helios, BAASBox e OpenMobster, destas optamos pela plataforma OpenMobster objeto de estudo da presente pesquisa, por ser, durante o desenvolvimento da pesquisa, a mais madura e com melhor documentação. A plataforma de MBaaS OpenMobster pode ser definida como uma plataforma *open source* que provê integração entre aplicações móveis e serviço de *cloud*. Seu objetivo é fornecer a infraestrutura para que o desenvolvedor de aplicações móveis possa construir produtivamente aplicativos que consumam informações que residem na nuvem. A plataforma fornece uma sincronização entre a aplicação e os dados alocados na nuvem através de API e *frameworks* que ficam do lado servidor para expor os serviço em nuvem; do lado do dispositivo é disponibilizado um conjunto de *frameworks* e APIs para o desenvolvimento de aplicativos móveis (OPENMOBSTER, 2014b), este fornece os seguintes recursos:

- *Data Synchronization*: consiste basicamente em manter o armazenamento na nuvem, sincronizando automaticamente sempre que o dispositivo estiver disponível, isso permite que trabalhe tanto *online* quanto *off-line*. Os dados são automaticamente sincronizados com o serviço de nuvem com base nas alterações locais do estado. Estas mudanças de estado são auto detectada e não requer nenhuma programação relacionada com sincronização do lado do dispositivo por parte do desenvolvedor.

- *Real-Time Push Notifications*: Alterações na nuvem são automaticamente empurradas para o dispositivo. O mecanismo de envio utiliza uma abordagem baseada puramente em rede/socket, em vez de metodologias que utilizam alertas de *Short Message Service* (SMS) ou *e-mail* de avisos. As notificações por *push* acontecem dentro do ambiente de execução do aplicativo

- *Mobile Remote Procedure Call (RPC)*: fornece os serviço através de um mecanismo de RPC.

- *Management Console*: fornece um console de gerenciamento para os dispositivos conectados a *mobile cloud*.

Para executar a plataforma OpenMobster é necessário a instalação do servidor *web* JBoss e o ambiente de execução Java. A plataforma pode prover diversos tipos de serviços desde sistemas que persistam informações em um Sistema gerenciador de banco de dados (SGBD) a serviço de autenticação *Lightweight Directory Access Protocol* (LDAP). No presente estudo foi abordado um sistema que persiste informações em um SGBD Hsqldb, bem como em uma base de dados MySQL. A arquitetura da plataforma pode ser observada na Figura 2.4, onde a plataforma de MBaaS OpenMobster faz interface com os clientes móveis, fornecendo a estes, serviços de *backend* que podem abranger diversas categorias, desde aplicações de *e-mail* e serviços *web* até mobilização de banco de dados.

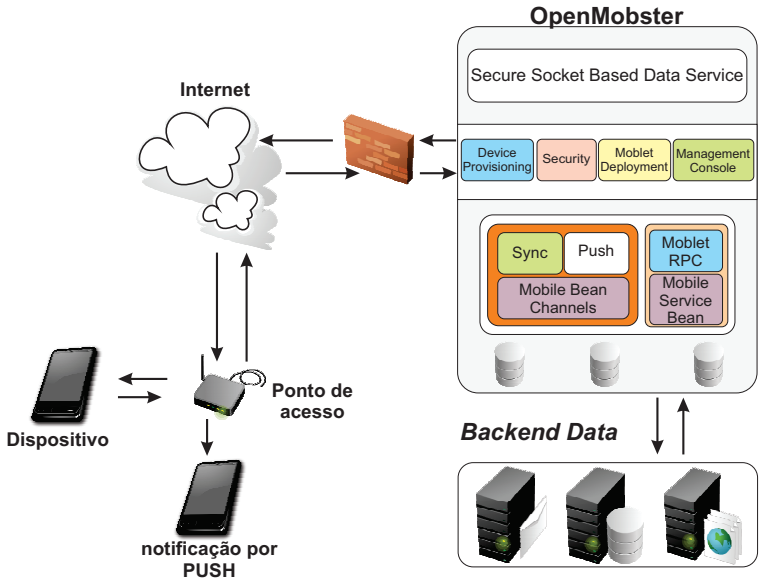


Figura 2.4: Arquitetura do OpenMobster

2.4 Dependabilidade

O conceito de dependabilidade não é atual. Surgiu pela primeira vez em 1830, no contexto da máquina de calcular de Babbage (AVIZIENIS; LAPRIE; RANDELL, 2001). Atualmente devido à prestação de serviços onipresente na Internet, a dependabilidade tornou-se um atributo de principal interesse em desenvolvimento de *hardware/software*, implantação e operação (MACIEL et al., 2011). A dependabilidade pode ser entendida como a capacidade de oferecer uma funcionalidade específica, que pode ser justificadamente confiável (AVIZIENIS et al., 2004), ou ainda, que o sistema irá executar ações especificadas ou apresentar resultados específicos de maneira confiável e em tempo oportuno (PARHAMI, 1988).

Um aplicativo requer atributos funcionais e não funcionais para garantir a satisfação e fidelidade dos usuários, assim, a dependabilidade é um conceito que engloba os seis principais requisitos não funcionais (AVIZIENIS; LAPRIE; RANDELL, 2001), listados abaixo:

- **Confiabilidade** - A probabilidade que o sistema irá prover o serviço continuamente,

sem erros, até um certo instante *t*;

- **Disponibilidade** - A capacidade do sistema estar de prontidão para prover o serviço corretamente;
- **Segurança** - Ausência de consequências catastróficas para o(s) usuário(s) e o ambiente;
- **Confidencialidade** - Ausência de divulgação desautorizada de informação;
- **Integridade** - Ausência de alterações impróprias no estado do sistema;
- **Manutenabilidade** - A habilidade de sofrer reparos e modificações;

Avaliação da dependabilidade está ligada ao estudo efeito de erros, defeitos e falhas no sistema; uma vez que estes provocam um impacto negativo nos atributos de dependabilidade (OLIVEIRA et al., 2013). Técnicas de prevenção, predição, remoção e tolerância a faltas, por sua vez, contribuem para manter níveis desejados de dependabilidade, sobretudo em sistemas críticos (AVIZIENIS; LAPRIE; RANDELL, 2001). Dessa forma, podemos definir **falha** como a falha de um componente, subsistema ou sistema que interage com o sistema em questão (MACIEL et al., 2011). Um **erro** é definido como um estado que pode levar a ocorrência de uma falha. Um **defeito** representa o desvio do funcionamento correto de um sistema. A dependabilidade está intimamente relacionada com as disciplinas de tolerância a falhas e confiabilidade (MELO et al., 2014). Dentre os requisitos apresentados destacamos a disponibilidade, que está relacionada a confiabilidade. Embora os conceitos de disponibilidade e confiabilidade estejam relacionados, a confiabilidade pode ser definida como a probabilidade de um sistema *S* não falhar até um tempo *t*, ou seja, a confiabilidade de um sistema no tempo *t*, é a probabilidade de o sistema executar até o tempo *t* sem que apresente falhas. A função da confiabilidade é representada por *R(t)*, onde *t* corresponde ao tempo que se deseja obter a confiabilidade (MACIEL et al., 2011). Matematicamente a confiabilidade pode ser vista através da Equação 2.1.

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

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

A disponibilidade pode ser definida como a quantificação da alternância entre a provisão correta e incorreta do serviço em questão (NICOL; SANDERS; TRIVEDI, 2004). Assim, podemos afirmar que na avaliação de disponibilidade é observado os eventos que podem levar o sistema a falhar, bem como os eventos de manutenção que fazem com que o sistema retorne ao seu funcionamento correto. Em outras palavras, a disponibilidade é utilizada para estimar a proporção de tempo que o sistema está ativo dentro de todo o seu tempo de vida (MACIEL et al., 2011). Também podemos definir a disponibilidade estacionária como a razão entre o tempo de funcionamento esperado e a soma dos tempos de funcionamento e falha esperados (MACIEL et al., 2011). A disponibilidade estacionária pode ser definida pela seguinte Equação:

$$A = \frac{E[Uptime]}{E[Uptime] + E[Downtime]} \tag{2.2}$$

Onde,

- *A é a disponibilidade estacionária do sistema*;
- $E[Uptime]$ *é o tempo de funcionamento esperado do sistema*;
- $E[Downtime]$ *é o tempo de falha esperado do sistema*;

Se o sistema se aproxima dos estados estacionários quando o tempo aumenta, é possível quantificar a disponibilidade de estado estacionário, de tal forma que é possível estimar a longo prazo a fração de tempo que o sistema está disponível (MACIEL et al., 2012).

$$A = \lim_{t \to \infty} A(t),\ \mathrm{t} \geq 0 \tag{2.3}$$

A disponibilidade é frequentemente representada em número de noves. O número de noves corresponde a contagem dos algarismos consecutivos iguais a 9, após a vírgula (MARWAH et al., 2010). Por exemplo, se um sistema apresenta estar disponível 99,9% do tempo, a sua disponibilidade pode ser representado por 3 noves. O número de noves são calculados através da Equação 2.4.

$$N = \log_{10}(1 - A) \tag{2.4}$$

Ao contrário da disponibilidade, a indisponibilidade representa a probabilidade do sistema não estar disponível, pode ser calculada a partir das Equações 2.5 e 2.6.

$$UA = \frac{E[Downtime]}{E[Downtime] + E[Uptime]} \tag{2.5}$$

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

Utilizando-se do valor da indisponibilidade é possível obter o *downtime* estimado do sistema em um intervalo de tempo. O *downtime* anual representa o número esperado de horas que o sistema estará indisponível no intervalo de um ano, é calculado pela fórmula da Equação 2.7.

$$Downtime_{anual} = UA \times 8760h \tag{2.7}$$

Quando os tempos de *uptime* e *downtime* não estão disponíveis, geralmente são usados os valores médios entre os eventos de falha e reparo do sistema. A Figura 2.5 mostra uma representação visual destes valores.

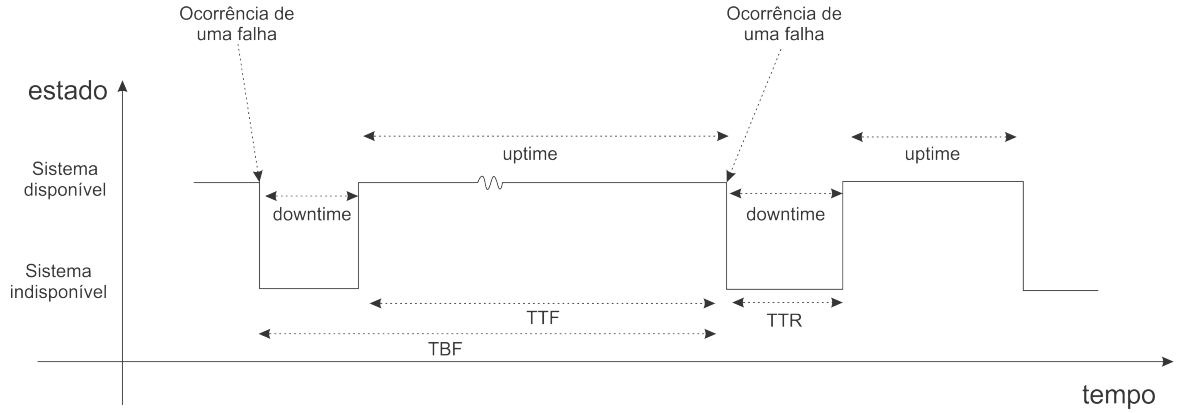


Figura 2.5: *Uptime* e *downtime*, adaptada de (MACIEL et al., 2011)

- **Tempo médio para a falha** - *Mean Time To Failure* (MTTF) – É o tempo médio para a ocorrência de falhas do sistema. Corresponde ao valor médio para a métrica TTF da Figura 2.5. O MTTF pode ser calculado através da Expressão 2.8;

$$MTTF = \int_0^\infty R(t) \times dt, \tag{2.8}$$

- **Tempo médio para reparo** - *Mean Time To Repair* (MTTR) – É o tempo médio para levar o sistema novamente ao estado de funcionamento, após a ocorrência de uma falha. Corresponde ao valor médio para a métrica TTR, na Figura 2.5. A obtenção do MTTR pode ser realizada pela Equação 2.9;

$$MTTR = \int_0^\infty M(t) \times dt. \tag{2.9}$$

Onde $M(t)$ é a função de manutenabilidade descrita na Equação 2.10.

$$M(t) = P(D < t),\ \mathrm{t} >= 0. \tag{2.10}$$

Onde D é uma variável aleatória que representa o tempo de reparo.

- **Tempo médio entre falhas** - *Mean Time Between Failures* (MTBF) – É o tempo médio entre a ocorrência de falhas. Corresponde ao valor médio para a métrica TBF da Figura 2.5.

A Equação para o cálculo da disponibilidade pode ser escrita em função do MTTF e do MTTR, conforme apresentado na Equação 2.11.

$$A = \frac{MTTF}{MTTF + MTTR} \tag{2.11}$$

Quando o MTTF é muito maior que o MTTR, a disponibilidade pode ser avaliada de acordo com a Equação 2.12.

$$A = \frac{MTBF}{MTBF + MTTR} \tag{2.12}$$

2.5 Modelos para Avaliação de Confiabilidade e Disponibilidade

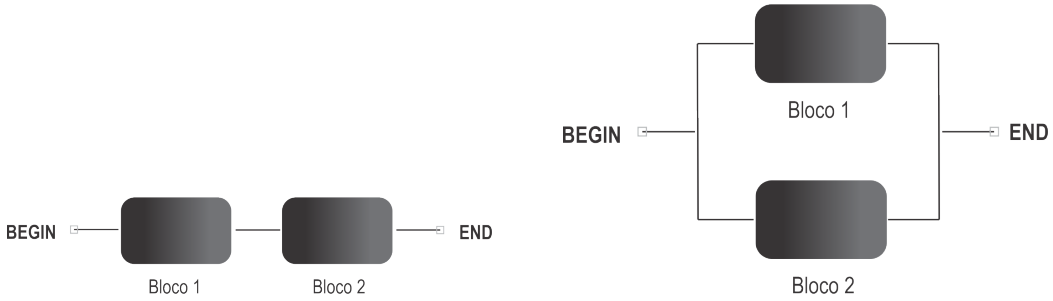
A modelagem analítica desempenha um papel importante na avaliação de dependabilidade. Modelos analíticos para avaliação da disponibilidade e avaliação da confiabilidade podem ser classificados em duas categorias: modelos combinatoriais e baseados em estados (MACIEL et al., 2012). Modelos combinatoriais representam o fracasso ou os modos operacionais por relações estruturais entre componentes e subsistemas que compõem o sistema completo. Exemplos de modelos combinatoriais são RBD e *Fault Trees* (O'CONNOR; O'CONNOR; KLEYNER, 2012). Modelos baseados em estados descrevem o comportamento do sistema em termos de estados e ocorrências de eventos, expressas como as transições de estado marcados. Cadeias de Markov (BOLCH et al., 1998) e SPN (MOLLOY, 1982) são dois tipos principais de modelos baseado em estados. Em cada uma das categorias de modelo existem vantagens e desvantagens (MALHOTRA; TRIVEDI, 1994). Modelos combinatoriais são mais simples e fáceis de criar, porém, devemos assumir que os componentes são estocasticamente independentes. Modelos baseados em estados são mais complexos e difíceis de criar, porém, estes possuem maior poder de expressividade comparado à modelos combinatoriais. Com modelos baseado em estados, podemos representar cenários mais complexos, como mecanismos de redundância *cold standby* ou *warm standby* (LOGOTHETIS; TRIVEDI, 1995).

Através de modelagem hierárquica é possível combinar as duas categorias de modelos, aproveitando assim o que cada uma tem de melhor (MALHOTRA; TRIVEDI, 1993). Nesta seção introduziremos conceitos básicos dos modelos utilizados nesta pesquisa.

2.5.1 Diagramas de Bloco de Confiabilidade

O modelo combinatorial Diagrama de Blocos de Confiabilidade (RBD) é um método utilizado para mostrar como a confiabilidade de um componente pode contribuir para o sucesso ou o fracasso de um sistema complexo. Em outras palavras, é um tipo de modelo proposto inicialmente para analisar a confiabilidade de sistemas baseados nas relações de seus componentes. Estes modelos apresentam um conjunto de blocos, que podem ser organizados em série, em paralelo ou em estruturas *k-out-of-n*, ou ainda em combinações entre essas organizações (TRIVEDI et al., 1996). O modelo RBD fornece uma iteração lógica entre componentes do sistema, definindo quais combinações ativas definem a funcionalidade do sistema. Com essa técnica é possível calcular também as métricas de dependabilidade, tais como, a disponibilidade e manutenabilidade. Um RBD é formado pelos seguintes componentes: vértice de origem, vértice de destino, componentes do sistema (representados por blocos) e arcos conectando os blocos e

os vértices. A Figura 2.6 traz alguns exemplos de RBD, apresentando estruturas básicas em série (Figura 2.6(a)) e em paralelo (Figura 2.6(b)).



(a) Série (b) Paralelo

Figura 2.6: Estrutura básica de um RBD

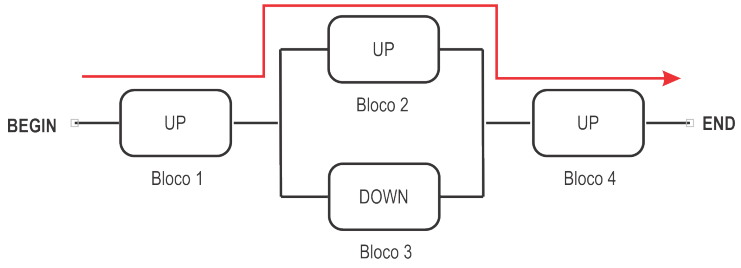
A estrutura do RBD apresenta a organização necessária para o sistema funcionar, ou seja, o diagrama representa o modo operacional do ambiente que está sendo modelado (MELO et al., 2013). O modo operacional apresenta quais componentes do sistema devem estar em funcionamento para que o sistema deva responder corretamente. Assim, podemos observar que se o sistema necessitar que todos os componentes estejam em funcionamento, para que o mesmo possa responder adequadamente, este modelo deve ser organizado em série. Caso seja necessário que ao menos um componente esteja funcionando, para o sistema responder, os blocos do modelo devem ser organizados em paralelo. Estruturas *k-out-of-n* representam situações onde, para o sistema funcionar é necessário que, ao menos k componentes funcionem dentro de um montante n (WEYNS; HÖST, 2013). Um sistema modelado em RBD está disponível caso haja um caminho contínuo do vértice de origem ao vértice de destino. Na Figura 2.7 demonstramos esse conceito, onde na Figura 2.7(a), demonstramos um sistema disponível, no qual é possível através do vértice de origem atingir o vértice de destino, e na Figura 2.7(b), é possível verificar um sistema indisponível, pois as falhas dos componentes impedem que através do vértice de origem, seja alcançado o vértice de destino.

A disponibilidade de modelos RBD com blocos em série é obtida através do produto das disponibilidades de cada um dos n blocos componentes (ČEPIN, 2011). Deste modo, a disponibilidade estacionária pode ser calculada a partir da Equação 2.13, onde, A_s (*Availability*) corresponde ao valor da disponibilidade do sistema e A_i a disponibilidade do componente i.

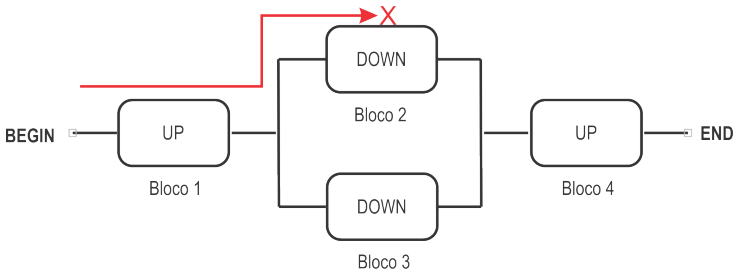
$$A_s = \prod_{i=1}^{n} A_i \tag{2.13}$$

Para uma estrutura em paralelo de n componentes independentes, a disponibilidade estacionária é dada por:

$$A_s = 1 - \prod_{i=1}^{n} (1 - A_i) \tag{2.14}$$



(a) Sistema disponível



(b) Sistema indisponível

Figura 2.7: Disponibilidade em RBD

Onde, A_s é a disponibilidade do sistema e A_i a disponibilidade do componente i, $(1-A_i)$ corresponde a indisponibilidade estacionária.

A organização em blocos permite a modelagem de situações onde os comportamentos de falha e reparo de cada componente não afetam os demais (MACIEL et al., 2011). Isso facilita a modelagem de sistemas que não possuem dependência entre os componentes. Dessa forma, caso seja necessário a modelagem de sistema mais complexos, onde exista uma dependência entre os componentes, o projetista deve recorrer a uma abordagem de modelagem hierárquica ou a um modelo de espaço-estados.

2.5.2 Cadeia de Markov de Tempo Contínuo

Modelos de Markov são blocos de construção fundamentais sobre os quais a maioria das técnicas quantitativas de análise de desempenho são construídas (TRIVEDI, 2002). Cadeias de Markov é um modelo matemático baseado em estados que pode ser utilizado para representar as interações entre os vários componentes do sistema, tanto para fins descritivos quanto preditivos (MENASCE; DOWDY; ALMEIDA, 2004). Este modelo foi proposto inicialmente por Andrei Andreevich Markov em 1907, e têm sido utilizados intensivamente na modelagem de confiabilidade e desempenho nos últimos cinquenta anos(MACIEL et al., 2011). Na ciência da computação, em particular, este formalismo é bastante conveniente para descrever e analisar propriedades dinâmicas de sistemas computacionais (BOLCH et al., 2001). Para entender o conceito de cadeia de Markov de tempo contínuo, primeiro é preciso apresentar a definição de processo estocástico. Um processo estocástico é definido por uma família de variáveis aleatórias $X_t : t \in T$, onde cada variável aleatória X_t é indexada por um parâmetro t e o conjunto de todos

os valores possíveis de X_t representa o espaço de estados do processo estocástico.

O parâmetro t pode ser definido como parâmetro de tempo, caso este satisfaça a condição: $T \subseteq \mathbb{R}^+ = [0, \infty)$. Se o conjunto T é discreto, o processo é classificado como de tempo discreto, caso contrário, é de tempo contínuo. Da mesma forma, o espaço de estados S pode ser discreto ou contínuo, dividindo processos estocásticos em dois grupos: de estado discreto e de estado contínuo. Se o espaço de estados é discreto, o processo estocástico é chamado de cadeia (MACIEL et al., 2011).

Assim, um processo estocástico de estado discreto é markoviano se, para todo $t_0 < t_1 < ... < t_n < t_{n+1}$ e para todo $X(t_0), X(t_1), ..., X(t_n), X(t_{n+1})$, os valores de $X(t_{n+1})$ dependerem somente do último valor $X(t_n)$ e não dos valores anteriores $X(t_0), X(t_1), ..., X(t_{n-1})$ (BOLCH et al., 2001), ou seja:

$$P(X_{t_{n+1}} \leq s_{n+1} | X_{t_n} = s_n, X_{t_{n-1}} = s_{n-1}, ..., X_{t_0} = s_0) = P(X_{t_{n+1}} \leq s_{n+1} | X_{t_n} = s_n) \quad (2.15)$$

Desta forma, esta propriedade define que, uma vez que um sistema esteja em um determinado estado, a transição para o próximo estado depende apenas do estado atual onde este se encontra. O histórico de estados que esteve anteriormente não importa e não é lembrado (SIEGERT; FRIEDRICH; PEINKE, 1998). Por isso, esta propriedade é também chamada de *memoryless property*.

A cadeia de Markov constitui um tipo particular de processo estocástico com estados discretos e com o parâmetro de tempo podendo assumir valores contínuos ou discretos (SOUSA; ROMERO MARTINS MACIEL, 2009). Assim, podemos dividir as cadeias de Markov em duas classes, de acordo com o parâmetro de tempo: *Discrete-Time Markov Chain* (DTMC), para tempo discreto e CTMC, para tempo contínuo. A principal diferença entre essas duas classes é que nas CTMCs, as transições entre os estados podem ocorrer em qualquer instante do tempo, enquanto que nas DTMCs, as transições só podem ser feitas em pontos discretos no tempo.

Graficamente uma cadeia de Markov pode ser representada como um diagrama de estado, onde os vértices representam os estados e os arcos do diagrama representam as possíveis transições entre os estados. As transições entre estados representam a ocorrência de eventos (MACIEL et al., 2011). A Figura 2.8 apresenta um exemplo de CTMC com dois estados e duas transições. Neste modelo o estado 1 representa que o sistema está funcionando. A partir deste a falha pode ocorrer (λ), levando ao estado 2. Do estado 2 apenas o reparo (μ) é possível.

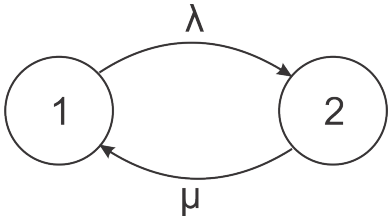


Figura 2.8: Exemplo de CTMC

Os pesos das transições possuem um significado que varia, caso a cadeia seja de tempo discreto ou contínuo. Em uma DTMC, o peso de um arco em um estado s_i para s_j corresponde a probabilidade de que, uma vez que o sistema se encontra no estado s_i, aconteça uma mudança para o estado s_j no próximo intervalo de tempo. Consequentemente, esse valor deve ser menor que 1 e a soma de todos os pesos de arcos que saem de um estado s_i também não deve exceder 1 (Oliveira, 2011).

Enquanto nas CTMCs, o peso do arco corresponde a taxa de migração de um estado s_i para o estado s_j. O inverso deste valor corresponde ao tempo médio de permanência no estado s_i. A propriedade de não guardar memória das cadeias de Markov implica que este tempo deve ser conduzido por uma distribuição com esta mesma propriedade (BOLCH et al., 1998). A taxa CTMC ou probabilidade DTMC de transição de estados do modelo ocorre obedecendo a uma lei exponencial ou geométrica, respectivamente (SOUSA; ROMERO MARTINS MACIEL, 2009).

Cadeias de Markov também possuem uma representação em forma de matriz, denominada matriz de taxa de transição Q. A matriz Q possui as informações sobre as transições dos estados na cadeia de Markov. Esta é utilizada para a resolução de cadeias de Markov. Cada elemento localizado fora da diagonal principal representa a taxa de ocorrência dos eventos que efetivam a transição dos estados do sistema. Os elementos contidos na diagonal principal são os valores necessários para que a soma dos elementos de cada linha seja igual a zero. Por exemplo, utilizando a CTMC da Figura 2.8, teremos a matriz da Equação 2.16.

$$Q = \begin{pmatrix} -\lambda & \lambda \\ \mu & -\mu \end{pmatrix} \tag{2.16}$$

As probabilidades de transição dos estados podem ser calculadas através da Equação 2.17.

$$p_{i,j}(s,t) = P(X(t) = j | X(s) = i) \tag{2.17}$$

A análise estacionária de uma cadeia de Markov consiste em encontrar a probabilidade de o sistema estar em um determinado estado, considerando um longo tempo de execução. Estas probabilidades são independentes do estado inicial do sistema e são representadas pelo vetor $\pi = [\pi_1, \pi_2, ..., \pi_n]$, onde π_i é a probabilidade estacionária para o estado i. Uma vez que a cadeia de Markov considerada, é uma cadeia ergódica, podemos encontrar as probabilidades estacionárias através do sistema linear formado pelas Equações 2.18 e 2.19 (BOLCH et al., 2001). Se em uma cadeia de Markov é possível alcançar qualquer estado a partir de qualquer estado, em um número n finito de passos, está cadeia é dita ergódica.

$$\pi Q = 0 \tag{2.18}$$

$$\sum_{i=1}^{n} \pi_i = 1 \tag{2.19}$$

Onde na Equação 2.18, Q é a matriz de estados e π (vetor de probabilidade) é o autovetor correspondente ao autovalor unitário da matriz de transição. É importante ressaltar que a soma dos elementos do vetor de probabilidade π deve ser igual a 1, ou seja, $\|\pi\| = 1$ (ARAÚJO, 2009). A utilização de CTMCs permite avaliações tanto de desempenho, quanto de disponibilidade. Suas aplicações vão desde a quantificação da vazão de uma linha de produção até a determinação de tempos de falha e reparo em sistemas críticos (BAIER et al., 2003).

2.6 Análise de Sensibilidade

A análise de sensibilidade é um termo genérico utilizado como técnica de classificação para determinar os fatores mais relevantes nas métricas de um modelo (HAMBY, 1994). Tal método pode ajudar identificando componentes importantes para melhorar a confiabilidade e o desempenho, como também identificar gargalos do sistema. Existem vários métodos disponíveis para a realização de análise de sensibilidade, entre estes podemos citar o projeto de experimento fatorial, análise de correlação, análise de regressão, análise de perturbação e análise diferencial, que também é conhecido como análise de sensibilidade paramétrica ou método direto (HAMBY, 1994; MATOS et al., 2012; MATOS; MACIEL, 2011). Análise diferencial, também conhecido como análise de sensibilidade paramétrica ou método direto, é a base de quase todas as outras técnicas de análise de sensibilidade (MATOS et al., 2012), (MELO et al., 2014). Há um consenso entre os autores que os modelos são realmente sensíveis a parâmetros de entrada de duas maneiras distintas: (1) a variabilidade, ou incerteza, de um parâmetro de entrada sensível tem uma grande contribuição à variabilidade de saída total; e (2) pode haver uma alta correlação entre um parâmetro de entrada e os resultados do modelo, o que significa dizer que pequenas mudanças nos valores de entrada resultam em mudanças significativas na saída (HAMBY, 1994).

Uma outra técnica de análise de sensibilidade é um dos métodos mais simples que consiste em variar um parâmetro por vez repetidamente, mantendo os outros fixos. Ao aplicar este método, o *ranking* de sensibilidade é obtido observando as mudanças correspondentes na saída do modelo. A análise de sensibilidade paramétrica pode ser utilizada em modelos analíticos, tais como cadeias de Markov, onde é uma técnica particularmente importante, para verificar o efeito causado sobre as métricas de interesse por mudanças nas taxas de transição. Dentre os vários benefícios da análise de sensibilidade, Matos Junior (2011) cita a identificação de parâmetros que podem ser removidos sem efeito significativo para os resultados. Modelos grandes, com dezenas de taxas, podem ser drasticamente reduzidos usando esta abordagem.

A análise diferencial foi escolhida para este trabalho, pois pode ser realizada de forma eficiente em modelos analíticos e é normalmente utilizada em estudos de disponibilidade e desempenho. Esta análise é realizada através do cálculo das derivadas parciais para as métricas de interesse de cada parâmetro. Por exemplo, considerando a métrica Y, que depende de um parâmetro λ, a sensibilidade de Y com relação a λ é calculado com a Equação 2.20 ou 2.21 quando é adotado a escala de sensibilidade (FRANK, 1978).

$$S_\lambda(Y) = \frac{\partial Y}{\partial \lambda} \tag{2.20}$$

$$S^*_\lambda(Y) = \frac{\partial Y}{\partial \lambda}\left(\frac{\lambda}{Y}\right) \tag{2.21}$$

Podem ser utilizados outros métodos de escala, dependendo da natureza dos parâmetros, a medida de interesse, e o requisito para remover os efeitos de unidades (Matos Junior, 2011).

$S_\lambda(Y)$ e $S^*_\lambda(Y)$ são também referidos como coeficientes de sensibilidade, (HAMBY, 1994), cujos valores ordenados produzem o *ranking* usado para comparar o grau de influência entre todos os parâmetros.

2.7 Injeção de Falhas

Ao se conduzir um experimento que tenha como objetivo a avaliação de dependabilidade, uma técnica importante a ser considerada, é a utilização de injeção de falha, pois falhas ocorrem de forma imprevisível dentro de um sistema, podendo levar muito tempo para a observação desse evento. A utilização de técnicas de injeção de falha auxilia no controle e monitoramento do experimento, observando o comportamento do sistema durante a ocorrência de eventos de falha. De acordo com (ARLAT et al., 1990), técnicas de injeção de falhas permitem a observação do comportamento do sistema através de experimentos controlados, nos quais a presença de falhas é induzida explicitamente nos componentes do sistema. Existem dois tipos de implementação para injetores de falhas: injetores de falha em nível de *hardware*, e injetores de falha em nível de *software* (ZIADE et al., 2004). As falhas de hardware podem ser de dois tipos: com contato e sem contato (HSUEH; TSAI; IYER, 1997). Na injeção de falha em *hardware* com contato, o injetor tem contato físico direto com o sistema alvo e pode produzir variações de corrente ou tensão no sistema testado. Na abordagem sem contato, o injetor não se conecta com o sistema alvo. Neste caso, para produzir as falhas, este produz algum evento externo que acarreta a falha do componente desejado. Exemplos deste tipo de evento: uma radiação intensa ou uma interferência eletromagnética (HSUEH; TSAI; IYER, 1997).

As técnicas de injeção de falhas por *software* também são divididas em duas categorias: em tempo de compilação ou em tempo de execução (HSUEH; TSAI; IYER, 1997). Injetores de falha em tempo de compilação fazem alterações do código fonte do sistema original, a fim de emular o efeito de falhas de *hardware* e *software* no sistema. Injetores de falhas de tempo de execução, por sua vez, não exigem a modificação do código fonte do sistema. Em vez disso, este tipo de injetor utiliza algum mecanismo que serve de gatilho para a injeção de falhas. Existem três técnicas para implementar o gatilho que irá disparar as falhas:

- ***Time-out*** - Falha simples inserida após a contagem de um tempo qualquer. Geralmente ocasiona uma interrupção no sistema. O contador utilizado pode ser implementado em *hardware* ou em *software*;

- ***Exceção/trap*** - Nesta técnica, quando uma certa exceção de *software* ou interrupção de *hardware* ocorre, o controle é transferido para o injetor de falhas. Esta técnica permite a injeção de falhas após a ocorrência de eventos específicos, ou seja, não há controle de tempo para execução de falhas;

- ***Inserção de código*** - Consiste em inserir linhas de código que ocasionam falhas antes de determinadas instruções. As instruções originais não são modificadas, em vez disso, o código de injeção de falhas é adicionado após certos pontos do programa em memória principal. Podemos afirmar que, neste caso, o injetor é parte do próprio *software* que está sendo testado.

A injeção de falhas no *hardware* exige a aquisição de equipamentos custosos, enquanto que a injeção de falhas em *software* não possui tal exigência, sendo menos onerosa. Também existe um risco de danificação permanente do dispositivo alvo, algo que não ocorre com a injeção de falhas em *software* (ZIADE et al., 2004). A presente pesquisa utilizou falhas injetadas em nível de *software* com a técnica *Time-out* que consiste na injeção de falhas ao final de um tempo determinado. De acordo com (OLIVEIRA et al., 2013), injetores em nível de *hardware* também possuem um número limitado de pontos de injeção das falhas, enquanto que os injetores de *software* possuem um número de pontos de injeção mais amplo.

Existem diversas ferramentas para injeção de falhas, tanto a nível de *software* como de *hardware*, por exemplo, (ZHANG; LIU; ZHOU, 2011), apresenta uma ferramenta de injeção de falhas para sistemas embarcados em tempo real chamados DDSFIS (*Debug-based Dynamic Software Fault Injection System*). (COOPER et al., 2010) apresenta uma ferramenta chamada YCSB (*Yahoo! Cloud Serving Benchmark*) que consiste em um benchmark para testes em bancos de dados com o apoio de computação em nuvem, unindo os dois conceitos. Contudo, podemos destacar o Eucabomber proposto inicialmente por (SOUZA et al., 2013), que é capaz de inserir eventos de falha e reparo no ambiente de computação em nuvem Eucalyptus. A abordagem de inserção utilizada é baseada em *software*, onde comandos para paralisação e *restart* de serviços são inseridos, baseados em variáveis aleatórias. A mesma ferramenta Eucabomber foi atualizada por (BRILHANTE et al., 2014) que estendeu a anterior, proporcionando novas métricas orientadas ao usuários, considerando as operações de ciclo de vida da VM, como também levando em conta as dependências de falha entre os componentes do Eucalyptus. Desta maneira, utilizando esta abordagem é possível verificar se os comportamentos descritos em modelos para o ambiente Eucalyptus são suficientemente aceitáveis como comportamento dos sistema real.

2.8 Intervalo de Confiança da Disponibilidade

A credibilidade de uma modelo analítico pode ser obtida através de um processo de validação, que tem como objetivo mostrar que um modelo obtido é compatível com o ambiente

pretendido. Existem diversas técnicas de validação de modelos, no entanto, o trabalho ora posto utilizou o método do intervalo de confiança da disponibilidade proposto por (KEESEE, 1965). Para efetuar o cálculo utilizando este método, é necessário inicialmente ter alguns dados como: tempo total de falhas e reparos, e quantidade de eventos aplicadas durante os testes para validação. Inicialmente a equação da disponibilidade estacionária é adaptada, conforme a Equação 2.22.

$$A = \frac{\mu}{\lambda + \mu} = \frac{1}{1 + \frac{\lambda}{\mu}} = \frac{1}{1 + \rho} \tag{2.22}$$

Onde ρ é relação entre $\frac{\lambda}{\mu}$. Ainda de acordo com KEESEE (1965), considera-se para o experimento o n sendo os eventos de falha e reparo, tempo total de falha sendo S_n e do tempo total de reparo é Y_n. Para estimar o valor de λ, é utilizada a Equação 2.23.

$$\hat{\Lambda} = \frac{n}{S_n} \tag{2.23}$$

Um intervalo de confiança $100 * (1 - \alpha)$ para λ é dado por uma distribuição Qui-quadrada com os parâmetros definido na Equação 2.24.

$$\left(\frac{\chi^2_{2n;1-\frac{\alpha}{2}}}{2S_n}, \frac{\chi^2_{2n;\frac{\alpha}{2}}}{2S_n}\right) \tag{2.24}$$

Para estimar μ, segue-se o mesmo processo usado em λ, conforme Equação 2.25.

$$\hat{M} = \frac{n}{Y_n} \tag{2.25}$$

E para o intervalo de confiança $100 * (1 - \alpha)$ para μ é definido pela Equação 2.26.

$$\left(\frac{\chi^2_{2n;1-\frac{\alpha}{2}}}{2Y_n}, \frac{\chi^2_{2n;\frac{\alpha}{2}}}{2Y_n}\right) \tag{2.26}$$

Dessa forma, o estimador de probabilidade máxima para a relação $\frac{\lambda}{\mu}$ é $\hat{\rho}$ que é definido na Equação 2.27.

$$\hat{\rho} = \frac{\hat{\Lambda}}{\hat{M}} = \frac{\frac{n}{S_n}}{\frac{n}{Y_n}} = \frac{Y_n}{S_n} \tag{2.27}$$

e um $100 \times (1 - \alpha)$ intervalo de confiança para ρ é dado para o (ρl, ρu), descrita por uma função de probabilidade da distribuição F.

$$\rho l = \frac{\hat{\rho}}{f2n;2n;\frac{\alpha}{2}} \text{ e } \rho u = \frac{\hat{\rho}}{f2n;2n;1-\frac{\alpha}{2}} \tag{2.28}$$

Assim, torna-se possível calcular o intervalo de confiança para a disponibilidade do sistema real, onde o estimador de probabilidade máximo para a disponibilidade é $\hat{A} = \frac{1}{1+\hat{\rho}}$. Uma vez que a disponibilidade A é uma função monotonicamente decrescente de ρ, o intervalo de

confiança $100 \times (1-\alpha)$ para A, ou seja, A_U e A_L, é encontrado através da Equação 2.29.

$$(\frac{1}{1+\rho u}, \frac{1}{1+\rho l}) \tag{2.29}$$

Com o intervalo de confiança definido, a validação do modelo proposto pode ser realizada, e pode, por conseguinte, ser determinado se a disponibilidade calculada através do modelo está dentro do intervalo de confiança obtido para o sistema real.

3

Metodologia e Arquiteturas MBaaS

Diversas arquiteturas com o intuito de prover um serviço estável podem ser propostas para a implantação de um serviço de nuvem, especificamente neste caso, uma infraestrutura baseada em uma plataforma MBaaS. A definição de técnicas e estratégias que permitam a avaliação desses ambientes é de suma importância para empresas que utilizam esse tipo de serviço, pois permitem que as mesmas projetem um ambiente que cumpra níveis satisfatórios de disponibilidade, minimizando assim a falha desse serviço na visão do usuário final. Este capítulo apresenta a metodologia utilizada nessa pesquisa, bem como a arquitetura básica e baseada em uma nuvem privada da plataforma MBaaS, que foram utilizadas para a elaboração dos principais modelos analíticos.

3.1 Metodologia

Esta seção apresenta a metodologia adotada para a avaliação e modelos de disponibilidade da plataforma de MBaaS, que consiste em seis atividades: entendimento da plataforma de MBaaS, definição da arquitetura básica, modelagem da arquitetura básica, experimento de injeção de falhas e monitoramento, refinar modelo gerado, comparação dos resultados e validação do modelo, e avaliação dos resultados e proposição de arquiteturas e cenários. Apenas aspectos gerais pertinentes à modelagem e avaliação de disponibilidade realizada foram abordados. Detalhes técnicos como o referente ao experimento serão melhor detalhados nas próximas seções. A Figura 3.1 apresenta o diagrama de atividades da metodologia adotada.

- **Entendimento da plataforma de MBaaS**: Esta etapa compreende a identificação das características da plataforma de MBaaS, bem como seus componentes e interações. Além de uma verificação minuciosa das principais funcionalidades da plataforma, também foi verificada nesta fase a aplicação que seria empregada para a realização dos experimentos, neste caso utilizou-se uma aplicação de criação de *ticket*, que é disponibilizada junto a plataforma OpenMobster como aplicação de teste. Nesta fase, é importante definir quais métricas de dependabilidade serão avaliadas no

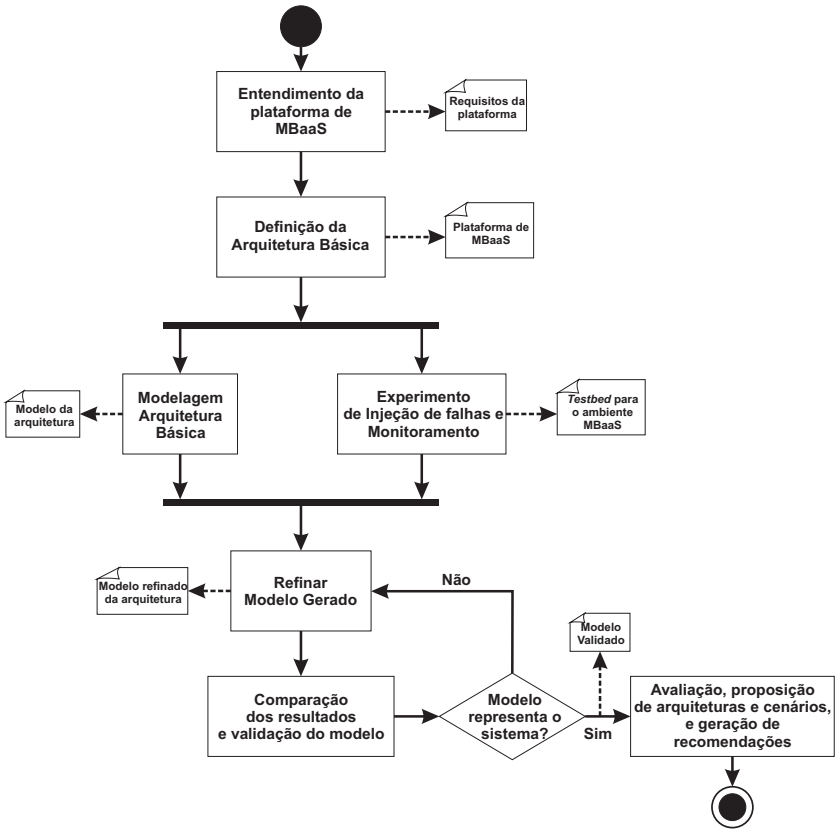


Figura 3.1: Metodologia aplicada

serviço em questão. No presente trabalho foram definidas as métricas de disponibilidade e indisponibilidade da plataforma. A partir desta fase é possível ter uma visão mais aprofundada da plataforma quanto aos seus componentes e funcionalidades, proporcionando assim subsídios para a definição das arquiteturas;

- **Definição da arquitetura básica**: A arquitetura básica foi definida como a arquitetura que possui os requisitos mínimos para o provimento do serviço, o intuito foi verificar os principais componentes do serviço sob o ponto de vista da disponibilidade no lado servidor, ou seja, os componentes que em caso de falha afetam significativamente a disponibilidade do sistema, esta fase é melhor detalhada na Seção 3.2;

- **Modelagem da arquitetura básica**: Com base na arquitetura básica foram propostos modelos analíticos hierárquicos com o intuito de descrever a disponibilidade do sistema, onde todos os componentes foram considerados, como *hardware*, sistema operacional, e os *softwares* que fazem parte da plataforma OpenMobster, que são a JVM, o JBoss, o OpenMobster, e o Banco de Dados. Nesse sentido, foram utilizados dois formalismos, para representar o relacionamento entre os componentes de alto nível do ambiente. Para o *hardware*, o sistema operacional e a plataforma MBaaS, utilizou-se o diagrama de blocos de confiabilidade (RBD), já para representar as

relações dos componentes que formam o bloco da plataforma de MBaaS, utilizamos o modelo baseado em estado CTMC;

Ainda nesta fase, são inseridos valores de falha e reparo nos parâmetros dos componentes dos modelos, esses valores foram obtidos através de experimentos, pesquisa em sites de fabricante e pesquisa em artigos relacionados. Após essas atividades é possível obter resultados de disponibilidade e indisponibilidade desses modelos;

- **Experimento de Injeção de falhas e Monitoramento**: Aqui, foi feita a montagem do ambiente de teste, onde é instalado e configurado o sistema conforme a arquitetura básica para a realização do experimento de injeção de falhas. Para isso, desenvolveu-se em *Shell Script* algoritmos que inserem falhas nos componentes do sistema com o objetivo de simular o comportamento do ambiente real, bem como foi criado um algoritmo com a mesma linguagem capaz de monitorar o serviço com o objetivo de verificar os estados do sistema, ou seja, se o mesmo está operacional ou não. Como os algoritmos de injeção de falhas injetam a falha ao final de um tempo exponencialmente distribuído, esses tempos correspondem às taxas de falha e reparos dos componentes do sistema, que são as mesmas taxas utilizadas nos modelos da fase anterior, este processo é mais bem detalhado no Capítulo 5. Com a execução do experimento é obtido métricas de disponibilidade do sistema, que serão utilizadas no processo de validação do mesmo;

- **Refinar modelo gerado**: Após a geração dos modelos analíticos é verificado se, de fato, estes estão corretos de acordo com o ambiente montado. Nesta fase é feito um refinamento dos modelos sob as mesmas condições do ambiente de testes;

- **Comparação dos resultados e validação do modelo**: A fim de verificar se os modelos analíticos propostos são suficientemente aceitáveis como modelos que representam o ambiente real, foi realizada a validação de tais modelos. Para isso os resultados gerados pelos modelos propostos foram analisados, e em posse destes, com os resultados obtidos no experimento de injeção de falhas, foi realizada a validação dos modelos, que consistiu em, a partir do experimento realizado na fase anterior, encontrar o intervalo de confiança da disponibilidade, para o qual foi utilizado o método proposto por (KEESEE, 1965) que será melhor descrito no Capítulo 5. Após essa fase é obtido um modelo confiável do sistema, ou seja, que represente o ambiente real o mais próximo possível;

- **Avaliação dos resultados e proposição de arquiteturas e cenários**: Com a arquitetura definida e os modelos que a representam validados, é possível realizar a avaliação dos resultados, como a análise de dependabilidade, verificando métricas de disponibilidade e indisponibilidade anual do ambiente proposto. Da mesma forma, também é possível a criação de cenários de interesse e proposição de arquiteturas, sendo para

isso necessário utilização de análise de sensibilidade, com o objetivo de encontrar gargalos de disponibilidade no ambiente. Como resultado dessa fase, na presente pesquisa foi possível a partir do modelo da arquitetura básica, propor novas arquiteturas, como por exemplo, modelos baseado em nuvem privada, mais especificamente a plataforma *Eucalyptus* com a utilização da plataforma de MBaaS, a qual o modelo de serviço foi validado. Destarte, também implementamos um cenário na arquitetura básica com um processo de recuperação automática na JVM, além de criar vários cenários variando a probabilidade de sucesso desse processo de recuperação.

A metodologia teve o objetivo principal de seguir os passos de forma sistemática para o obtenção dos resultados da presente pesquisa.

3.2 Arquitetura Básica

A arquitetura básica foi baseada na plataforma de MBaaS OpenMobster, cujo principal objetivo foi através dela propor um modelo de disponibilidade da plataforma. Na Figura 3.2 exibimos uma representação dessa arquitetura, que é formada por três componentes principais, o hardware, o sistema operacional e o sistema OpenMobster que pode ser subdividido em mais quatro componentes: a *Java Virtual Machine*, o banco de dados, o servidor web e a própria plataforma de MBaaS.

Nesta arquitetura a plataforma de MBaaS foi implantada sobre um servidor web, e o serviço provido para monitoramento foi um sistema simples de *ticket*, que é um sistema de teste padrão da plataforma OpenMobster, sendo importante enfatizar que o serviço e a plataforma representam o mesmo componente. O servidor web utilizado pela plataforma é o Jboss (JBOSS, 2014), que é um servidor de aplicações baseado em Java, dessa forma é necessário que a *Java Virtual Machine* esteja instalado no servidor, pois sem ela o servidor Jboss não é capaz de ser executado. Vale ressaltar que a JVM é um dos componentes mais críticos do sistema do ponto de vista da disponibilidade, pois o mesmo encontra-se na base da pilha de *software* e uma vez esse componente estando indisponível, todos os componentes a cima dele também ficaram indisponíveis.Por este motivo, propomos uma variação do modelo concebido para essa arquitetura, onde foi implementado um processo de recuperação automática na JVM. O banco de dados utilizado nesse ambiente foi o *Hyper Structured Query Language Database* (HSQLDB) (HSQLDB, 2014), que é um banco de dados relacional escrito em Java. Todos esses componentes são importante do ponto de vista da disponibilidade no lado servidor, visto que a falha em um deles acarreta na indisponibilidade do serviço. Na seguinte seção é apresentada uma arquitetura baseada em nuvem privada que tem o intuito de aumentar a disponibilidade do serviço, e que utilizam as funcionalidades da plataforma aqui apresentada.

Quanto ao modo operacional desse ambiente, entende-se que a plataforma estará indisponível quando ocorrer alguma falha no *hardware*, no sistema operacional ou na plataforma.

Uma falha no *hardware* derruba todo o sistema tornando o serviço da plataforma indisponível. Da mesma forma, uma falha no sistema operacional ocasionará na indisponibilidade do serviço, bem como um defeito na plataforma, que pode ocorrer em qualquer um dos componentes de *software* que formam a pilha do serviço ilustrada na Figura 3.2.

Dessa forma, o modo operacional para a plataforma pode ser sintetizado na Expressão Lógica 3.1 (onde o *Hardware*, o Sistema Operacional e a Plataforma MBaaS são representados por HW, SO e PM, respectivamente).

$$Plataforma_{UP} = HW_{UP} \wedge SO_{UP} \wedge PM_{UP} \tag{3.1}$$

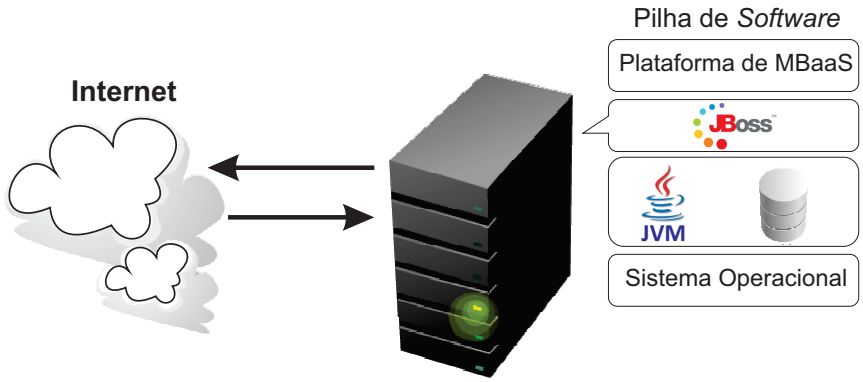


Figura 3.2: Arquitetura Básica

3.3 Arquitetura Baseada em Nuvem Privada

Essa arquitetura foi baseada na plataforma Eucalyptus, a partir dela foram propostos modelos de disponibilidade para plataforma de MBaaS em nuvem privada. A Figura 3.3 apresenta a arquitetura que é formada pelos componentes, cliente e o serviço de nuvem. O cliente com dispositivo *mobile*, pode se conectar ao serviço através da internet, no entanto, como o principal objetivo dessa pesquisa é analisar o lado servidor, a figura apresenta em mais detalhes a infraestrutura de nuvem privada, onde o serviço é provido basicamente, como já descrito na Subseção 3.2, por um servidor **web**, pela JVM e por um banco de dados. O componente banco de dados é instalado na máquina *Frontend* e fica disponível para conexão com a plataforma de MBaaS. A JVM, o servidor *web* e a plataforma de MBaaS são instalados em uma máquina virtual, que por sua vez é instanciada pelos recursos físicos do controlador do Nó, que é executado em cada Nó físico e controla o ciclo de vida das instâncias em execução no Nó (DANTAS et al., 2012). É importante ressaltar que a infraestrutura de nuvem baseada na plataforma Eucalyptus é composta de mais componentes, como por exemplo, a máquina *Frontend* é composta por *Hardware*, sistema operacional, Controlador da Nuvem (CLC), o Controlador do cluster (CC), o Controlador de armazenamento (SC) e o Walrus. Já a máquina Nó é composta por *Hardware*, Sistema Operacional, KVM, que é o *hypervisor* em execução no Nó, e o Controlador do Nó (NC).

Na Seção 4.2 apresentamos o modelo de disponibilidade para esta arquitetura, e detalhamos cada componente da mesma que influencia na disponibilidade do sistema. Também no Capítulo 4, apresentamos uma análise de sensibilidade aplicada nesse ambiente, que teve como objetivo encontrar gargalos de disponibilidade, assim foram propostos mais três cenários que são descritos detalhadamente nas Seções 4.4, 4.5 e 4.6.

O modo operacional dessa arquitetura compreende que, para o sistema ficar indisponível é necessário que ocorra algum defeito em algum dos componentes que a compõe, como por exemplo: na plataforma de MBaaS, na VM, no *Frontend*, banco de dados ou no Nó. Uma falha na VM ocasionará na indisponibilidade do serviço, assim como um defeito na plataforma de MBaaS que está instalada na VM. A principal função do Nó é controlar o ciclo de vida das instâncias, desta maneira, problema neste componente interrompe o serviço, visto que a VM é executada por ele. Já uma falha no banco de dados impede que os dados na nuvem sejam sincronizados com os dispositivos móveis, impossibilitando assim execução normal do serviço. Como o *Frontend* é responsável por realizar operações de gerenciamento na nuvem, quando se torna defeituoso, perde-se o controle de acesso às VMs, tornando assim o serviço inoperante.

Como a plataforma de MBaaS está instalada na VM, representamos esses dois componentes como único, e nomeamos de Subsistema MBaaS. Assim sendo, o modo operacional para essa arquitetura pode ser representado pela Expressão Lógica 3.2, onde (o *Frontend*, o Nó, o Banco de Dados e o Subsistema MBaaS, são representados FE, NO, BD e SM respectivamente).

$$Plataforma_{UP} = FE_{UP} \wedge NO_{UP} \wedge BD_{UP} \wedge SM_{UP} \tag{3.2}$$

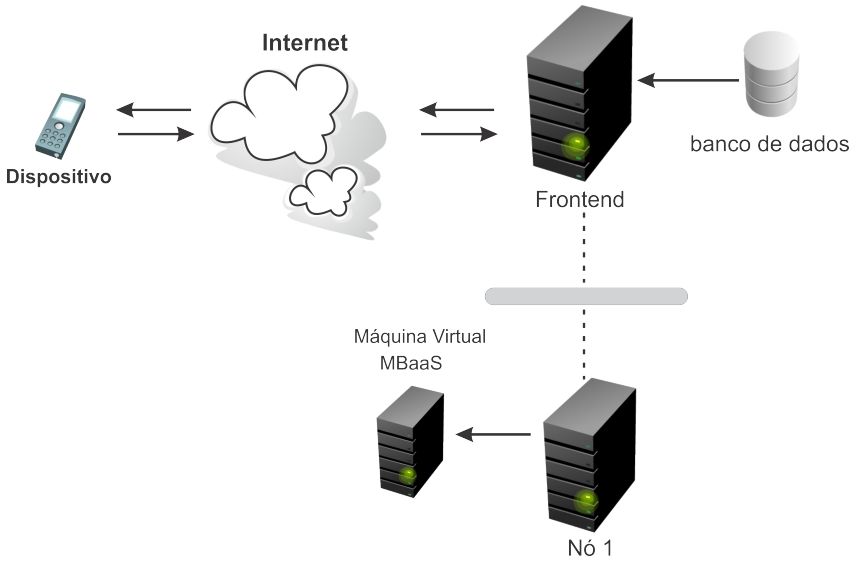


Figura 3.3: Arquitetura nuvem privada

3.4 Considerações Finais

Este capítulo apresentou a metodologia adotada para a avaliação do ambiente MBaaS, detalhando seus passos e expondo os principais produtos de cada etapa, bem como foi apresentada as duas principais arquiteturas trabalhadas durante essa pesquisa. Primeiramente foi apresentada a arquitetura básica, que consistiu em apresentar a plataforma de MBaaS em um ambiente mais simples sem componentes redundantes, neste ambiente apenas uma máquina provia o serviço através de um servidor *web*. Na outra arquitetura já foi apresentada uma infraestrutura de nuvem privada, porém, também sem componentes redundantes. O próximo capítulo apresenta os modelos analíticos para as arquiteturas aqui propostas.

4

Modelos para as Arquiteturas propostas

Neste capítulo, apresentamos os modelos criados para a avaliação da disponibilidade das arquiteturas propostas. Inicialmente é descrito o modelo e detalhado os componentes da arquitetura básica, que foi introduzida na Seção 3.2. Apresentamos também uma variação dele com um processo de recuperação automática na JVM, que tem como objetivo o aumento da disponibilidade estacionária. Na Seção 4.2, apresentamos o modelo para a arquitetura baseada em nuvem sem redundância, apresentada na Seção 3.3, bem como a análise de sensibilidade aplicada nessa arquitetura. Na Seção 4.4, apresenta o modelo para arquitetura baseada em nuvem com redundância no *Frontend*. A Seção 4.5, é apresentado o modelo da arquitetura baseada em nuvem com redundância aplicada nos Nós, e na Seção 4.6 é apresentado o modelo com redundância em todos os componentes, Nós e *Frontend*. Todos os modelos redundantes baseado em nuvem visam alcançar melhores níveis de disponibilidade estacionária do serviço.

4.1 Modelos para Arquitetura Básica

Esta seção descreve os modelos de disponibilidade da arquitetura básica apresentada na Seção 3.2, que foram concebidos a fim de representar o comportamento da plataforma de MBaaS OpenMobster, no qual o modo operacional é descrito na Expressão Lógica 3.1 localizada na Seção 3.2. Foi representado também um cenário com o processo de recuperação automática no componente JVM. A quantificação da disponibilidade de sistemas complexos de TI pode ser alcançado através da representação de estados do sistema com a modelagem hierárquica de RBDs e cadeias de Markov (KIM; MACHIDA; TRIVEDI, 2009a). Portanto, foi utilizada a técnica de modelagem hierárquica heterogênea para representar a plataforma de MBaaS, onde modelos foram combinados sendo eles: *Reliability Block Diagrams* (RBD) e o modelo de espaço de estados *Continuos Time Markov Chain* (CTMC). Devido à complexidade dos elementos de *software* que compõe a plataforma de MBaaS, foi usado a modelagem com CTMC, pois de acordo com (MACIEL et al., 2011) para modelar interações mais complexas entre os componentes, são utilizados modelos estocásticos como cadeias de Markov ou mais geralmente modelos de espaço de estados. Os elementos representados nessa modelagem foram

a *Java Virtual Machine*, o JBoss, o OpenMobster e o Banco de Dados. Para representar o sistema completo, ou seja, o *Hardware*, o Sistema Operacional e o subsistema OpenMobster, foi empregado o modelo combinatorial RBD que tem o objetivo de representar a relação de confiabilidade entre os subsistemas.

Partindo do conceito que a utilização de modelos RBD permitem a modelagem de situações onde os comportamentos de falha e reparo de cada componente não afetam os demais (MACIEL et al., 2011), ou seja, representa a independência dos componentes modelados, dessa forma, podemos considerar que para cada componente da arquitetura básica, é atribuído a uma equipe específica de manutenção, isto é, para o componente *Hardware* uma equipe específica, assim como para os componentes Sistema Operacional e subsistema OpenMobster. No entanto, o componente subsistema OpenMobster é constituído por mais quatro componentes, deste modo e neste contexto, fica viável a modelagem desse bloco em CTMC, pois assim, apenas uma equipe ficaria responsável por esse bloco, e não quatro. Outro aspecto importante a ressaltar sobre a utilização de um modelo de espaço de estados CTMC na modelagem do serviço é sobre o cenário com processo de recuperação automática na JVM, onde para representá-lo é necessária à utilização de um modelo que demonstre a dependência entre os componentes. Nesse sentido, um modelo de espaço de estados é ideal. Um modelo CTMC é necessário devido à interdependência entre os componentes do subsistema, e significa também que uma equação de fórmula fechada de disponibilidade de estado estacionário pode ser obtida, a qual é útil para a computação de medidas desejadas (MELO et al., 2014). Assim, na presente pesquisa a métrica desejada foi a disponibilidade, que posteriormente será considerada no modelo de alto nível RBD que representa o sistema completo.

Os sete estados indicados na Figura 4.1 representam a plataforma de MBaaS OpenMobster. A notação adotada define o estado de cada componente, ou seja, (U) para *UP* e (D) para *DOWN*, e tem a seguinte sequência, o primeiro caractere representa o estado do componente banco de dados, o segundo da JVM, o terceiro do Jboss e o quarto caractere representa o estado do componente OpenMobster.

Os estados são *UUUU*, *UUUD*, *UUDD*, *UDDD*, *DDDD*, *DUUD* e *DUDD*, onde o estado *UUUU* representa a disponibilidade do serviço, e os outros seis estados representam a indisponibilidade do serviço.

A probabilidade do sistema estar disponível é a probabilidade de estar no estado *UUUU*, ou seja, $A_s = \pi_{(UUUU)}$, onde $\pi_{(UUUU)}$ é a probabilidade de estado estacionário estar no estado *UUUU*. No estado *UUUU* (Banco de Dados up, JVM up, JBoss up e OpenMobster up), o sistema está ativo, caso aconteça uma falha no OpenMobster com uma taxa λ_O o sistema vai para o estado *UUUD* (Banco de Dados up, JVM up, JBoss up e OpenMobster down), onde o sistema está em estado de quebra ou inativo, a partir desse estado a plataforma OpenMobster pode ser reparada com uma taxa μ_O retornando para o estado *UUUU*, ficando o sistema ativo novamente. Considerando uma falha no componente JBoss com uma taxa λ_B, o modelo é direcionado para o estado *UUDD* (Banco de Dados up, JVM up, JBoss down e OpenMobster

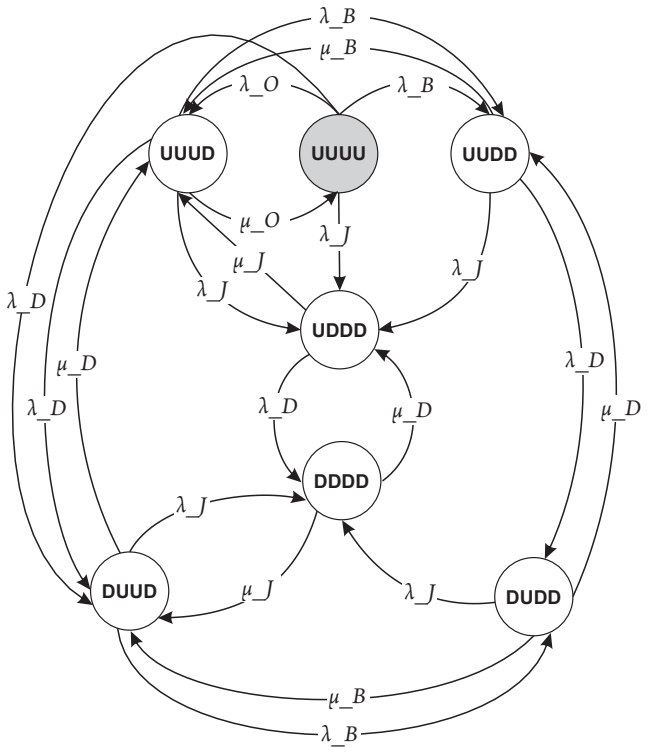


Figura 4.1: Cadeia de Markov da plataforma OpenMobster

down), onde o sistema também encontra-se em um estado inativo, desse estado pode acontecer um reparo no JBoss com taxa μ_B voltando para o estado *UUUD*. Desse estado pode ocorrer uma falha no componente JVM com uma taxa λ_J, indo para o estado *UDDD* (Banco de Dados up, JVM down, JBoss down e OpenMobster down). Dos estados *UUUU* e *UUUD* também pode ser disparada uma falha na JVM com taxa λ_J. Do estado *UUUU* pode suceder uma falha no banco de dados com taxa λ_D o sistema é então direcionado para o estado *DUUD* (Banco de Dados down, JVM up, JBoss up e OpenMobster down), uma vez nesse estado pode haver uma recuperação do banco de dados com taxa μ_D, retornando ao estado *UUUD*, como também ainda do estado *DUUD* pode haver uma falha no JBoss levando ao estado *DUDD* (Banco de Dados down, JVM up, JBoss down e OpenMobster down) desse estado pode acontecer um reparo do JBoss com taxa μ_B, retornando assim ao estado anterior *DUUD*, também do estado *DUDD* pode acontecer um reparo do banco de dados com taxa μ_D retornando ao estado *UUDD*, ainda do estado *DUUD* pode ocorrer uma falha na JVM com taxa λ_J indo para o estado *DDDD*.

O modelo de disponibilidade da plataforma pode ser avaliado utilizando a fórmula representada na Equação 4.1. Devido aos parâmetros λ_J, λ_B e λ_O ter valores iguais, na equação são representados pelo parâmetro λ. O mesmo se aplica aos parâmetros μ_J, μ_B e μ_O, que são representados pelo parâmetro μ.

$$A_{Sistema} = \frac{\mu^2 \mu_D}{(2\lambda + \mu)(3\lambda + \lambda_D + \mu)(\lambda_D + \mu_D)} \tag{4.1}$$

O modelo RBD (ilustrado na Figura 4.2) representa a relação entre os componentes individuais de todo o sistema: Hardware, o Sistema Operacional (SO) e o Sub-sistema MBaaS (que foi modelado separadamente com CTMC). Se pelo menos um dos componentes falhar, o sistema tornar-se-á indisponível, portanto, o RBD que o representa deve ser ligado em série. Da mesma forma, uma falha do sistema operacional faz com que o serviço da plataforma OpenMobster fique indisponível, e a falha na plataforma OpenMobster derruba todo o serviço. Assim, o modelo proposto pode ser utilizado para avaliar a disponibilidade para a plataforma OpenMobster em sua arquitetura básica e com processo de recuperação automática. O bloco de Sub-sistema MBaaS é representado através da disponibilidade do subsistema OpenMobster que foi obtida a partir dos modelos CTMCs mostrados nas Figuras 4.1 e 4.3.

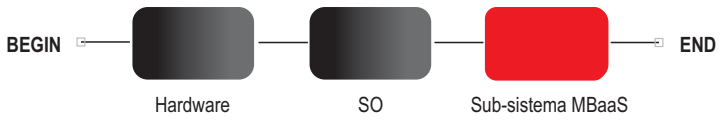


Figura 4.2: Modelo RBD de alto nível de todo o sistema

A disponibilidade de todo o sistema pode ser calculada através da Equação 4.2. Já a indisponibilidade do sistema pode ser calculada com a Equação 4.7, e o *downtime*, pode ser representado através da Equação 4.8.

$$A_{Sistema} = A_{HW} \times A_{SO} \times A_{SM} \tag{4.2}$$

Onde:
$A_{Sistema}$ representa a disponibilidade do Sistema;
A_{HW} representa a disponibilidade do *Hadware*;
A_{SO} representa a disponibilidade do Sistema Operacional;
A_{SM} representa a disponibilidade do Serviço MBaaS.

$$UA = 1 - A_{Sistema} \tag{4.3}$$

$$Downtime_{Sistema} = UA \times 8760h \tag{4.4}$$

4.1.1 Arquitetura Básica com Processo de Recuperação Automática

Considerando a hierarquia dos componentes de *software*, foi verificado durante os experimentos que a JVM é um dos componentes de mais baixo nível na pilha de *software* da

Figura 3.2, pois uma falha nesse componente torna o serviço indisponível, derrubando o JBoss e o OpenMobster. Dessa forma, propomos implantar um processo de recuperação automática nesse componente, a fim de aumentar a disponibilidade estacionária. Portanto, para representar o modelo com processo de recuperação automática foi utilizado a CTMC da Figura 4.1, onde dois estados foram adicionados UDDD Detecção Falha e DDDD Detecção Falha. A CTMC que representa a plataforma com o processo de recuperação automática é ilustrada na Figura 4.3.

Desta maneira, uma vez estando no estado *UDDD* ou *DDDD*, pode ocorrer o processo de recuperação automática, pois a JVM é o componente de mais baixo nível do sistema, uma falha nele, como já foi anteriormente explicado, torna todo o sistema indisponível, com exceção do banco de dados que está no mesmo nível da JVM. Uma falha nesse componente dispara o processo de recuperação automática, a taxa para executar esse processo $f\mu\_J$ multiplicado por F, onde $f\mu\_J$ é a taxa para o processo de recuperação automática da JVM e F é a probabilidade de sucesso desse processo, do estado *UDDD* o processo de recuperação automática direcionará para o estado *UUUD*, pois inicialmente é restabelecido o Jboss, apenas para após, ser reparado o OpenMobster, e do estado *DDDD* o processo de recuperação automática sendo executado com sucesso conduzirá para o estado *DUUD*, pois o banco de dados encontra-se no mesmo nível da JVM, sendo os dois independentes entre si. Caso o processo de recuperação automática não ocorra o sistema poderá ir para os estados $UDDD\_Detection\_Failure$ ou $DDDD\_Detection\_Failure$ com uma taxa $(1-F)\cdot f\mu\_J$, esses estados representam a falha do processo. A partir desses estados pode ocorrer um reparo manual na JVM com taxa μ_J, caso o sistema esteja no estado $UDDD\_Detection\_Failure$ o reparo na JVM o levará ao estado *UUUD*, caso esteja no estado $DDDD\_Detection\_Failure$ o reparo o irá direcionar ao estado *DUUD*.

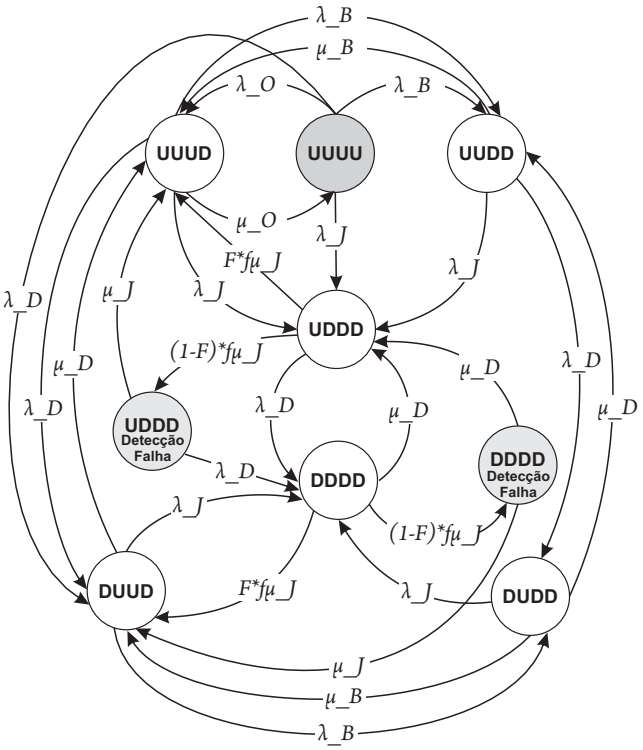


Figura 4.3: Cadeia de Markov da plataforma OpenMobster com o processo de recuperação automática

A disponibilidade do modelo com o processo de recuperação automática pode ser calculada utilizando a fórmula detalhada na Equação 4.5. A disponibilidade do sistema com o processo de recuperação automática pode ser calculada com a Equação 4.6.

$$A_{SMProc} = \frac{(\mu\mu_D(\alpha^2\beta_1 + \beta\mu(2\lambda^3\beta_2 + \beta_3) + \beta_4 + \alpha_{11}) + \alpha_{10})}{(\alpha_4(\alpha_5 + \alpha_6 + \beta(\lambda^2\alpha_7 + \alpha_8 + \alpha_9)))}, \tag{4.5}$$

Onde:

$\beta = F \cdot f\mu_J$,
$\beta_1 = (\lambda + \mu)(\lambda_D + \mu)(2\lambda + \lambda_D + \mu + \mu_D)$,
$\beta_2 = (\mu + \mu + D) + \beta(\lambda + \mu)(\lambda + \lambda_D + \mu + \mu_D)(2\lambda + \lambda_D + \mu + \mu_D)$,
$\beta_3 = \mu(\lambda_D + \mu + \mu_D)(\lambda_D^2 + \lambda_D\mu + \mu_D(\mu + \mu_D))$,
$\beta_4 = \lambda^2(\lambda_D^2 + 3(\mu + \mu_D)^2 + \lambda_D(3\mu + 2\mu_D))$,
$\beta_5 = \lambda_D^3 + \lambda_D^2$,
$\beta_6 = (4\mu + \mu_D) + \lambda_D(4\mu^2 + 3\mu\mu_D + \mu_D^2)$,
$\beta_7 = (\mu + \mu_D)(\mu^2 + 4\mu\mu_D + \mu_D^2)$,
$\beta_8 = (\lambda + \mu)(\lambda_D + \mu)(\lambda + \lambda_D + \mu_D)(\mu + \mu_D)(2\lambda + \lambda_D + \mu + \mu_D)$,
$\beta_9 = (\lambda_D + \mu) + 2\mu(\lambda_D + \mu)(\mu + \mu_D)(\lambda_D + \mu + \mu_D)$,
$\alpha = (1 - F) \cdot f\mu_J$,
$\alpha_1 = \lambda(\lambda_D + \mu)(\lambda_D(3\mu + 2\mu_D))$,

$\alpha_2 = (\mu + \mu_D)(7\mu + 2\mu_D)$,
$\alpha_3 = \lambda^2(2\lambda_D^2 + 4\lambda_D(2\mu + \mu_D) + \mu(7\mu + 5\mu_D))$,
$\alpha_4 = (2\lambda + \mu)(3\lambda + \lambda_D + \mu)(\lambda_D + \mu)(2\lambda + \lambda_D + \mu + \mu_D)$,
$\alpha_5 = (\alpha + \lambda)(\lambda_D + \mu)(\alpha + \lambda + \lambda_D + \mu_D)(\mu + \mu_D)$,
$\alpha_6 = \beta^2(\lambda + \mu)(\lambda + \lambda_D + \mu + \mu_D)$,
$\alpha_7 = (\lambda_D + 2\mu + \mu_D) + \lambda(\lambda_D + \mu + \mu_D)(\lambda_D + 2\mu + \mu_D)$,
$\alpha_8 = \mu(\lambda_D(\lambda_D + \mu) + \mu\mu_D + \mu_D^2)$,
$\alpha_9 = \alpha(2(\lambda_D + \mu)(\mu + \mu_D) + \lambda(\lambda_D + 2\mu + \mu_D))$,
$\alpha_{10} = \alpha(\beta_8 + \beta(2\lambda^3\beta_9 + \alpha_1 + \alpha_2 + \alpha_3))$, and
$\alpha_{11} = \lambda(\beta_5\beta_6 + \beta_7)$.

$$A_{SistemaProc} = A_{HW} \times A_{SO} \times A_{SM_{Proc}} \tag{4.6}$$

Onde:

$A_{SistemaProc}$ representa a disponibilidade do Sistema com processo de recuperação automática;
A_{HW} representa a disponibilidade do *Hadware*;
A_{SO} representa a disponibilidade do Sistema Operacional;
$A_{SM_{Proc}}$ representa a disponibilidade do Serviço MBaaS com processo de recuperação automática.

$$UA = 1 - A_{SistemaProc} \tag{4.7}$$

$$Downtime_{SistemaProc} = UA \times 8760h \tag{4.8}$$

É importante salientar que o principal objetivo desses modelos foi representar as transições de falha e reparo entre os componentes Banco de dados, JVM, JBoss e a plataforma de MBaaS OpenMobster, assim como também o processo de recuperação automática no componente *Java Virtual Machine*. A CTMC da Figura 4.1 não tem indício para ser refutada, pois a mesma passou por um processo de validação, que será melhor descrito no Capítulo 5.

4.2 Modelos para Arquitetura Baseada em Nuvem sem Redundância

Nesta seção apresentamos o modelo para a arquitetura ilustrada na Seção 3.3, que mostra os detalhes dos componentes de uma nuvem privada.

O modelo proposto aqui foi baseado na plataforma *Eucalyptus* (EUCALYPTUS, 2014), no entanto, os modelos podem ser adaptados para utilização em outras infraestruturas de nuvem privada, como OpenNebula, OpenStack, entre outras. O modelo é caracterizado por uma série

de componentes onde a falha de qualquer um destes provoca a indisponibilidade do lado da nuvem. O modo operacional para essa arquitetura é descrito na Seção 3.3, através da Expressão Lógica 3.2. É possível verificar a infraestrutura de nuvem privada com todos os componentes nos RBDs das Figuras 4.4, 4.5 e 4.6. A arquitetura da Figura 3.3 é representada através de modelos RBDs e CTMCs, estes modelos foram combinados constituindo um modelo hierárquico heterogêneo, o modelo RBD de alto nível da Figura 4.4 representa a arquitetura baseada em nuvem privada em uma visão *Top-Down*. Este modelo é dividido em quatro submodelos ligados em série, são eles: o *Frontend*, o Nó, o *Network-Attached Storage* (NAS) que é o componente de armazenamento o qual é criado por recursos físicos da máquina *Frontend*, mais especificamente pelo Controlador de Armazenamento, e o Sub-sistema que representa a plataforma OpenMobster. O *Frontend* é o nó físico que serve como ponto de acesso dos clientes a nuvem, como também executa os *softwares* de gerenciamento da infraestrutura de nuvem. O Nó é o componente que instancia e executa as VMs na infraestrutura de nuvem. O NAS, como já supracitado, é o componente de armazenamento e o Sub-sistema MBaaS que é a plataforma de MBaaS, a qual é melhor representada por uma CTMC, pelos mesmo motivos já descritos na seção anterior, pois é proposto na VM a implementação de um processo de recuperação automática na referida plataforma.

Figura 4.4: RBD da arquitetura baseada em nuvem privada

O submodelo *Frontend* é descrito na Figura 4.5. Este modelo consiste de *Hardware* (HW), Sistema Operacional (OS), e os subsistemas da plataforma Eucalyptus: Controlador da Nuvem (CLC), Controlador do cluster (CC), Controlador de armazenamento (SC) e o Walrus. Onde o CLC tem como principal função ser o ponto de acesso dos clientes na nuvem, este subsistema é também responsável por expor e administrar os recursos subjacentes virtualizados (servidores, rede e armazenamento) (SUNMICROSYSTEMS, 2009). O CC reúne informações sobre um conjunto de máquinas virtuais e horários de execução de VM no Controlador do Nó específico (DANTAS, 2013). O SC fornece armazenamento de bloco persistente para uso pelas instâncias de máquinas virtuais. Ele implementa acesso a bloco de armazenamento em rede, semelhante ao proporcionado pela *Amazon Elastic Block Storage* - EBS (EUCALYPTUS, 2009). Já o Walrus é um arquivo baseado em serviço de armazenamento de dados, sendo sua interface compatível com a *Amazon's Simple Storage Service* (S3) (EUCALYPTUS, 2009). Qualquer falha em um desses subsistemas irá tornar o *Frontend* inativo, pois todos estão ligados em série. O modo operacional do componente *Frontend* pode ser representado através da Expressão Lógica 4.9.

$$Frontend_{UP} = HW_{UP} \wedge SO_{UP} \wedge CLC_{UP} \wedge CC_{UP} \wedge SC_{UP} \wedge Walrus_{UP} \quad (4.9)$$

Figura 4.5: RBD do Frontend

A Equação 4.10 mostra a fórmula da disponibilidade para o submodelo *Frontend*.

$$A_{Frontend} = A_{HW} \times A_{SO} \times A_{CLC} \times A_{CC} \times A_{SC} \times A_{Walrus} \quad (4.10)$$

O submodelo Nó está ilustrado na Figura 4.6, e também é composto por *Hardware* (HW) e Sistema Operacional (OS), além dos componentes do Eucalyptus o *Hypervisor* que neste caso é o KVM, e o Controlador do Nó (NC), que são necessários para tornar o serviço disponível (DANTAS et al., 2012). O KVM é a camada de virtualização, enquanto que o Controlador do Nó controlam a execução, fiscalização e terminação das instâncias de VMs no *host* onde está sendo executado (DANTAS, 2013). A Expressão Lógica 4.11 representa o modo operacional deste componente.

$$No_{UP} = HW_{UP} \wedge SO_{UP} \wedge KVM_{UP} \wedge NC_{UP} \quad (4.11)$$

Figura 4.6: RBD do Nó

A Equação 4.12 apresenta a fórmula da disponibilidade para o submodelo Nó.

$$A_N = A_{HW} \times A_{SO} \times A_{KVM} \times A_{NC} \quad (4.12)$$

O submodelo Subsistema MBaaS foi modelado com a utilização de uma CTMC representada na Figura 4.7, isso devido a sua complexidade, onde é representada a instanciação da VM com a plataforma de MBaaS OpenMobster instalada. A partir dessa CTMC é possível realizar o cálculo dos valores de disponibilidade, que posteriormente serão utilizados pelo modelo de alto nível RBD da arquitetura de nuvem, bem como é possível obter uma fórmula fechada de disponibilidade de estado estacionário.

A CTMC da Figura 4.7 é composta pelos seguintes componentes: a plataforma OpenMobster e a VM. A CTMC é composta por 4 estados *UU*, *DU*, *DD* e *FDU*. Assim como na CTMC da arquitetura básica a notação adotada define os estados dos componentes, onde (U)

representa o estado *UP* e (D) representa o estado *DOWN*, a seguinte sequência foi adotada, o primeiro caractere está relacionado a plataforma de MBaaS, o segundo caractere está relacionado a VM, vale salientar que a plataforma de MBaaS já foi modelada com a CTMC da Figura 4.1. Os círculos brancos da CTMC da Figura 4.7 representam os estados onde o sistema está inativo, ou seja, onde o sistema está indisponível devido a uma falha. O círculo cinza indica o estado onde o sistema está ativo, ou seja, a probabilidade do sistema está ativo é a probabilidade de estar no estado *UU*. Apenas o estado *UU* representa a disponibilidade do serviço.

Do estado *UU* pode se alcançar os seguintes estados; a plataforma de MBaaS OpenMobster pode falhar com uma taxa λ_Open alcançando assim o estado *DU*, ainda do estado *UU* pode acontecer uma falha na VM com uma taxa λ_VM alcançando o estado *DD*. Do estado *DU* pode ser alcançado o estado *UU* com uma taxa $P \cdot \mu\_OpenM$, onde μ_OpenM é a taxa para recuperação automática da plataforma e *P* é a probabilidade de sucesso dessa recuperação acontecer, também do estado *DU* pode ser alcançado o estado *FDU* que representa a detecção de falha do processo de recuperação automática com uma taxa $(1-P) \cdot \mu\_OpenM$, onde $(1-P)$ representa a probabilidade desse processo falhar. Ainda do estado *DU* pode acontecer uma falha na VM e alcançar o estado *DD* com taxa λ_VM. A partir do estado *FDU* pode ser alcançado o estado *UU* com uma taxa μ_Open que representa a inicialização do serviço de forma manual. Do estado *DD* pode ocorre a instanciação da VM com a plataforma MBaaS já implantada com uma taxa μ_VM retornando para o estado que o sistema encontra-se ativo *UU*.

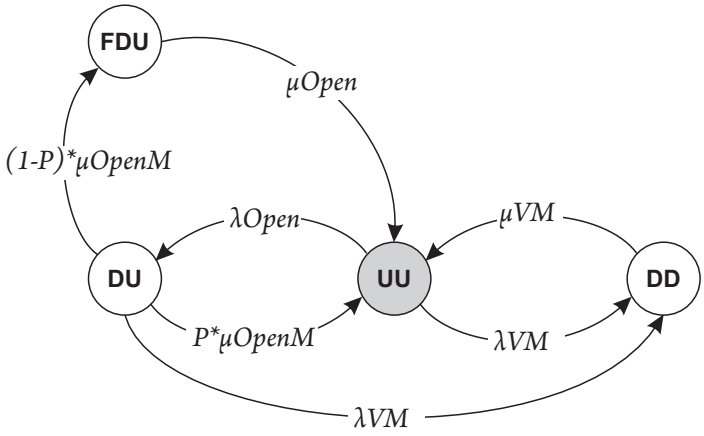


Figura 4.7: CTMC do Serviço

As taxas λ_Open e μ_Open foram extraídas do modelo CTMC da Figura 4.1 e representam a falha e reparo respectivamente da plataforma de MBaaS OpenMobster. As taxas de falha e reparo podem ser definida como $MTTF_i = \frac{1}{\lambda_i}$, e $MTTR_i = \frac{1}{\mu_i}$ se λ_i e μ_i forem constantes (MELO et al., 2014). A taxa μ_VM é a taxa de instanciação da VM somado ao tempo de inicialização da plataforma de MBaaS. A fórmula da Equação 4.13 foi obtida através da CTMC da Figura 4.7.

$$A_{servico} = \frac{\mu open(\lambda vm + \mu openM)\mu vm}{\lambda open\lambda vm\mu open + \lambda open(\mu open + \mu openM - P\mu open)\mu vm + \mu open\alpha\beta} \quad (4.13)$$

Onde:
$\alpha = (\lambda vm + \mu openM)$,
$\beta = (\lambda vm\mu vm)$,

O resultado da fórmula fechada pode ser inserido na Equação 4.14, a fim de obter o cálculo da disponibilidade de todo o sistema. Nesta Equação $A_{Frontend}$, A_N, A_{NAS} e $A_{servico}$, correspondem respectivamente a disponibilidade do *Frontend*, a disponibilidade do Nó, do NAS e a disponibilidade da plataforma MBaaS OpenMobster. O cálculo da indisponibilidade da plataforma de nuvem pode ser observado na Equação 4.23, já o *downtime* pode ser calculado através da Equação 4.24.

$$A_{nuvem} = A_{Frontend} \times A_N \times A_{NAS} \times A_{servico} \quad (4.14)$$

Onde:
A_{nuvem} representa a disponibilidade da nuvem privada com a plataforma de MBaaS;
$A_{Frontend}$ representa a disponibilidade do *Frontend*;
A_N representa a disponibilidade do Nó;
A_{NAS} representa a disponibilidade do NAS;
$A_{servico}$ representa a disponibilidade da plataforma de MBaaS.

$$UA_{nuvem} = 1 - A_{nuvem} \quad (4.15)$$

$$Downtime_{nuvem} = UA_{nuvem} \times 8760h \quad (4.16)$$

4.3 Análise de Sensibilidade Paramétrica para a Arquitetura Baseada em Nuvem sem Redundância

Em geral, os diferentes componentes que constituem um sistema computacional não necessariamente contribuem igualmente para a disponibilidade e desempenho do sistema (MATOS et al., 2012). Conhecer bem esse sistema e identificar quais são os elementos relevantes é fundamental para a métrica que está sendo avaliada, no caso disponibilidade. Nesse sentido a análise de sensibilidade surge para auxiliar na identificação desses componentes importantes para o sistema (MELO et al., 2014).

Desta forma, a fim de guiar a aplicação de redundância na arquitetura de nuvem privada

apresentada anteriormente, a análise de sensibilidade paramétrica foi utilizada, pois esta é uma abordagem que pode ser usada para encontrar gargalos de disponibilidade do sistema, orientando, assim, a melhoria e otimização (BLAKE; REIBMAN; TRIVEDI, 1988). Para realizar a análise de sensibilidade paramétrica dessa arquitetura foi necessário seguir alguns passos, que são sucintamente descritos da seguinte maneira:

Passo 1. Primeiramente definimos o modelo que será analisado, este foi um modelo RBD oriundo da arquitetura de nuvem privada sem redundância ilustrada na Figura 3.3 do Capítulo 3;

Passo 2. Logo após a definição do modelo, a ferramenta Mercury (SILVA et al., 2013), (SILVA et al., 2015) foi utilizada na análise de sensibilidade paramétrica, calculando os índices de sensibilidade para os modelo RBD, contudo, o mesmo cálculo pode ser realizado com a Equação 4.17;

$$\begin{aligned} S_\theta(A) = & \frac{\partial A_F}{\partial \theta} \times A_N \times A_{NAS} \times A_{Sistema} + \\ & A_F \times \frac{\partial A_N}{\partial \theta} \times A_{NAS} \times A_{Sistema} + \\ & A_F \times A_N \times \frac{\partial A_{NAS}}{\partial \theta} \times A_{Sistema} + \\ & A_F \times A_N \times A_{NAS} \times \frac{\partial A_{Sistema}}{\partial \theta} \end{aligned} \tag{4.17}$$

Onde:

A_F representa a disponibilidade do *Frontend*

A_N representa a disponibilidade do *Nó*

A_{NAS} representa a disponibilidade do *Banco de Dados*

$A_{Sistema}$ representa a disponibilidade da Plataforma MBaaS

As taxas de falha e de reparo dos componentes *Frontend*, Nó, NAS e Sub-sistema MBaaS são λ e μ respectivamente, as expressões derivadas correspondentes são apresentadas nas Equações 4.18 e 4.19. Onde o x representa os componentes individuais da arquitetura de nuvem privada sem redundância.

$$\frac{\partial A_x}{\partial \lambda_x} = -\frac{\mu_x}{(\mu_x + \lambda_x)^2} \tag{4.18}$$

$$\frac{\partial A_x}{\partial \mu_x} = -\frac{\mu_x}{(\lambda_x + \mu_x)^2} + \frac{1}{\lambda_x + \mu_x} \tag{4.19}$$

Dessa forma, os parâmetros de entrada λ e μ de cada componente são listados na Tabela 4.1, onde os valores são substituídos nas equações 4.18 e 4.19, a fim de obter os índices de sensibilidade que estão listados em um *Ranking* na Tabela 4.2, os valores de falha e reparo do *Frontend* (MTTFFrontend e MTTRFrontend) foram adquiridos de (DANTAS et al., 2012) e

(KIM; MACHIDA; TRIVEDI, 2009b), assim como os valores de falha e reparo do Nó. Os valores de falha e reparo do NAS foram verificados em (TEOREY; NG, 1998), enquanto os valores de falha e reparo do componente Sub-sistema MBaaS (MTTFSys e MTTRSys) foram obtidos da CTMC da Figura 4.7.

Tabela 4.1: Parâmetros de entrada

Parâmetros	**Valores** (h^{-1})
MTTFFrontend	180,72
MTTRFrontend	0,96999
MTTFNó	481,83
MTTRNó	0,91000
MTTFNAS	1440
MTTRNAS	3
MTTFSys	76,0337 h
MTTRSys	0,7275 h

Tabela 4.2: *Ranking* de sensibilidade dos componentes da arquitetura de nuvem privada

Parâmetros	**Valores**
MTTRFrontend	-5,338709083532889 10^{-3}
MTTFFrontend	5,338709083532851 10^{-3}
MTTFNAS	2,0790020790022045 10^{-3}
MTTRNAS	-2,0790020790020783 10^{-3}
MTTFNó	1,8850727099476226 10^{-3}
MTTRNó	-1,8850727099473832 10^{-3}
MTTFSys	2,2997001959954553 10^{-4}
MTTRSys	-2,2997001959951702 10^{-4}

Passo 3. Após realizado todos os cálculos é possível fazer uma análise dos resultados obtidos. Desta forma, podemos verificar que o componente *Frontend* tem os maiores índices sensitivos, sendo apontado como um gargalo na disponibilidade do sistema. Podemos observar ainda que o componente NAS também tem um impacto considerável na disponibilidade do sistema, no entanto, como esse componente na arquitetura considerada na Seção 3.3 é criado com os recursos físicos do *Frontend*, é possível salientar a importância da redundância neste componente. Ainda verificando os valores sensitivos do *Ranking* da Tabela 4.2, o componente Nó é outro que, podemos aplicar uma redundância para aumentar a disponibilidade do serviço, sendo assim fica evidente que a aplicação de mecanismos redundantes nos componentes da arquitetura de nuvem privada pode ocasionar no aumento da disponibilidade da mesma. As próximas seções apresentam os modelos das arquiteturas com a utilização de mecanismos redundantes nos componentes da arquitetura de nuvem privada.

4.4 Modelos para Arquitetura com Redundância no *Frontend*

A partir da análise de sensibilidade aplicada à arquitetura de nuvem sem redundância, foi verificado que o componente *Frontend* tem maior criticidade na disponibilidade da plataforma de MBaaS. Dessa forma, podemos afirmar que mudanças nesse componente influenciarão na disponibilidade do sistema, assim a Figura 4.8 representa a arquitetura que tem implementado dois *Frontend* da nuvem, utilizando o mecanismo de redundância *warm standby* com o objetivo de melhorar a disponibilidade da plataforma de MBaaS. Neste modelo, é proposta a utilização de dois *Frontends* que chamaremos de F1 e F2, no qual um está ativo, enquanto o outro encontra-se em estado *warm standby*, caso o *Frontend* que está ativo venha a apresentar uma falha o *Frontend* em *warm standby* detecta a falha e assumi a função do *Frontend* que estava ativo. A arquitetura do *Frontend* já foi melhor detalhada na Seção 4.2 através do modelo RBD da Figura 4.5. A arquitetura baseada em nuvem com a redundância no *Frontend* é melhor detalha no RBD da Figura 4.9, que contém três subsistemas, o *Frontend* que é o bloco que será aplicado à redundância, o subsistema Nó que já foi descrito nas seções anteriores e o sub-sistema MBaaS (SM), o NAS que é o dispositivo de armazenamento está composto nos *Frontends* redundantes. O processo de redundância dos *Frontends* é melhor representado através de uma CTMC, para isso utilizaremos o modelo CTMC da Figura 4.10 proposto por (DANTAS et al., 2012), este modelo representa o comportamento da redundância com mecanismo *warm standby* no *Frontend*. A Expressão Lógica 4.20 representa o modo operacional deste ambiente.

$$Plataforma_{UP} = (F1_{UP} \vee F2_{UP}) \wedge NO_{UP} \wedge SM_{UP} \tag{4.20}$$

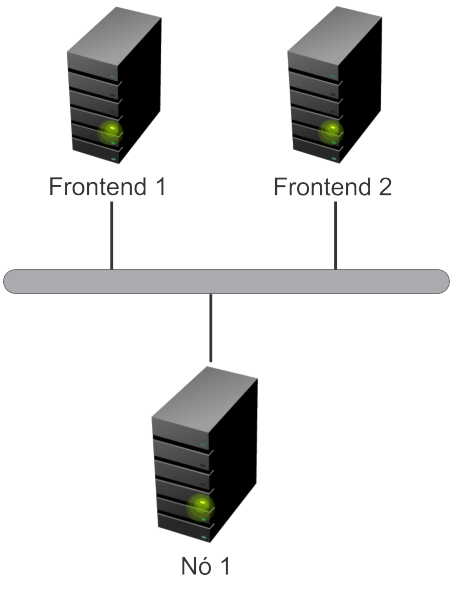


Figura 4.8: Arquitetura com redundância no *Frontend*

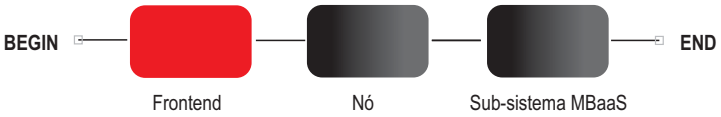


Figura 4.9: RBD com redundância no *Frontend*

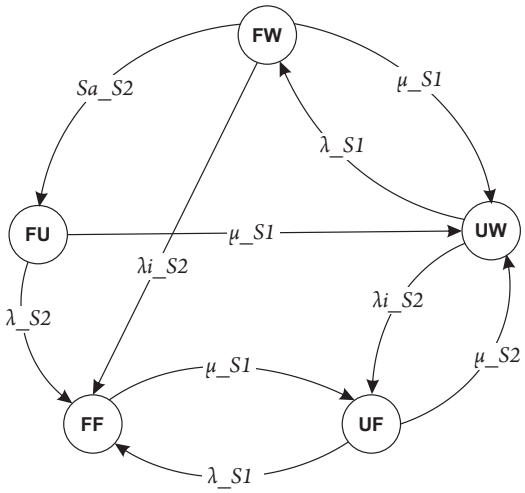


Figura 4.10: CTMC para redundância no *Frontend*

A CTMC da Figura 4.10 é composta por cinco estados: *UW*, *UF*, *FF*, *FU* e *FW*. Nos estados *UW*, *UF* e *FU* representam a disponibilidade do serviço, enquanto os estados *FF* e *FW* representam a indisponibilidade do sistema. No estado *UW*, o *Frontend* principal (F1) está ativo e o *Frontend* secundário (F2) está em *warm standby*. Se acontecer uma falha no F1, o sistema entra no estado *FW*, onde o *Frontend* secundário ainda não detectou a falha de F1. Caso o sistema secundário entre em falha o sistema pode sair do estado *FW* (estado que representa a falha do F1 e o F2 em espera) e ir para o estado *FF* (representando a falha dos dois servidores), a taxa de falha do *Frontend* secundário em espera é 20% menor do que o índice de falha de um *Frontend* ativo, já que o mesmo encontra-se ligado, mas não operacional. *FU* representa o estado onde F2 sai da condição de espera e assume o papel de *Frontend* ativo, enquanto F1 está em estado de falha. Se F2 falhar antes da reparação de F1 , o sistema vai para o estado *FF*.

De acordo com (DANTAS et al., 2012) a fim de dar prioridade à reparação do servidor principal, como política de manutenção, há apenas uma única passagem de reparação de *FF*, que vai para *UF*. Se o F2 falhar quando F1 está ativo, o sistema vai para o estado *UF*, retornando ao estado *UW* com o reparo do F2, ou vai para o estado *FF* em caso de o *Frontend* principal também falhar.

As taxas de falha dos *Frontends* são indicadas por λ_S_1 e λ_S_2, respectivamente. A taxa de $\lambda_i\_S_2$ representa a taxa de falha do servidor secundário quando ele está inativo. A taxa de reparo de *Frontend* secundário é de μ_S_2. A taxa de transição s_a_s2 representa a taxa de passagem, ou seja, o inverso do tempo médio para ativar o servidor secundário após uma falha no *Frontend* primário.

Através da CTMC podemos obter a fórmula algébrica para o cálculo da disponibilidade do bloco *Frontend* do RBD da Figura 4.9, como mostrado na Equação 4.21.

$$A_{Frontend\_reduntante} = \frac{\mu(\lambda \times \lambda_i + (\lambda_i + \mu)^2 + sa(\lambda + \lambda_i + \mu))}{sa(\lambda^2 + \lambda(\lambda_i + \mu) + \mu(\lambda_i + \mu)) + (\lambda + \mu)(\lambda \times \lambda_i + (\lambda_i + \mu)^2)}, \quad (4.21)$$

Para o cálculo da disponibilidade do sistema completo $A_{SistemaF}$ com a redundância nos *Frontends*, representado no RBD da Figura 4.9 pode ser utilizada a expressão algébrica demonstrada na Equação 4.22. A indisponibilidade do sistema pode ser representado pela expressão algébrica da Equação

$$A_{nuvem\_redundante} = A_{Frontend\_reduntante} \times A_N \times A_{servico} \quad (4.22)$$

$$UA_{nuvem\_redundante} = 1 - A_{nuvem\_redundante} \quad (4.23)$$

$$Downtime_{nuvem\_redundante} = UA_{nuvem\_redundante} \times 8760h \quad (4.24)$$

4.5 Modelos para Arquitetura com Redundância no Nó

Na análise de sensibilidade aplicada na arquitetura de nuvem sem redundância, o componente Nó aparece como o terceiro componente mais crítico no ambiente do ponto de vista da disponibilidade. No entanto, alterações nesse componente podem impactar na disponibilidade do sistema. Deste modo, propomos uma arquitetura com redundância nos Nós, que tem a infraestrutura ilustrada na Figura 4.11. O mecanismo de redundância aplicado aos Nós foi *warm standby*, onde o primeiro Nó encontra-se ativo e recebendo carga de trabalho, enquanto o segundo Nó encontra-se ligado, contudo, em um estado ocioso e sem receber carga de trabalho, por este motivo a probabilidade deste Nó falhar é menor. Caso o primeiro Nó venha a falhar, o segundo Nó ficará ativo, o serviço será reinicializado tornando, portanto, o sistema ativo novamente. O RBD da Figura 4.12 descreve o modelo para essa arquitetura. O modelo é composto por três subsistemas: o *Frontend* (FE) que já foi melhor descrito na Seção 4.2, o NAS que como já explicado na Seção 4.2 representa o armazenamento de dados e é criado pelos recursos físicos do *Frontend* e o bloco sub-sistema MBaaS, que neste caso é representado pela plataforma MBaaS mais a infraestrutura dos Nós que foi descrita na Seção 4.2 com o mecanismo de redundância *warm standby*. Devido as características deste bloco ele foi refinado utilizando uma CTMC. O modo operacional desta arquitetura pode ser representado através da Expressão Lógica 4.25.

$$Plataforma_{UP} = FE_{UP} \wedge NAS_{UP} \wedge (NO1_{UP} \vee NO2_{UP}) \quad (4.25)$$

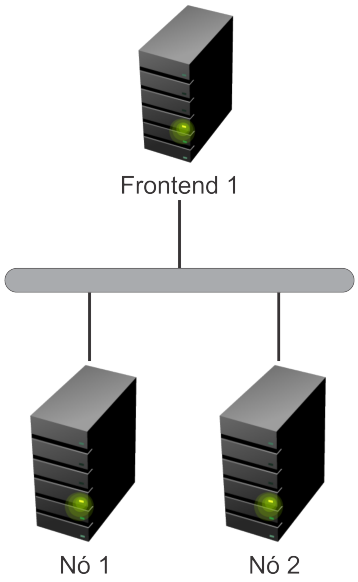


Figura 4.11: Arquitetura com redundância no Nó

Figura 4.12: RBD para arquitetura redundante nos Nós

Uma característica importante da CTMC da Figura 4.13 é a política de manutenção de priorização do serviço, ou seja, sempre que ocorrer uma falha no serviço será dado prioridade ao reparo do serviço, caso o Nó em *warm standby* esteja em estado de falha, e o serviço também esteja em estado de falha, porém, o Nó principal ou o secundário esteja ativo, a prioridade de reparo será dada ao serviço, apenas após esse reparo, ou caso aconteça uma falha nesse processo é que o Nó *warm standby* em estado de falha será reparado, e caso o serviço esteja em falha e um dos Nós estiver ativo e o outro estiver em estado *warm standby*, antes do chaveamento será dado prioridade a restabelecer o serviço no Nó ativo, se acontecer uma falha nesse procedimento, então será realizado o chaveamento.

A CTMC da Figura 4.13 compreende quinze estados, são eles: *UUW*, *UUD*, *UDU*, *UWU*, *DUD*, *DUW*, *DDW*, *DWD*, *DWU*, *DDU*, *DDD*, *FDUD*, *FDUW*, *FDWU* e *FDDU*. Os estados sombreados *UUW*, *UUD*, *UDU*, *UWU* representam a disponibilidade do serviço, enquanto os demais estados representam o serviço em falha ou a indisponibilidade do serviço. A notação adotada representa os estados de cada componente, (U) para o estado *UP*, (D) para o estado *DOWN* e (W) para o estado *warm standby*, a sequência adotada para os caracteres foi a seguinte, o primeiro caractere representa o estado da plataforma de MBaaS, o segundo representa o Nó principal com a infraestrutura descrita na Seção 4.2 mais a VM, o terceiro representa o Nó secundário.

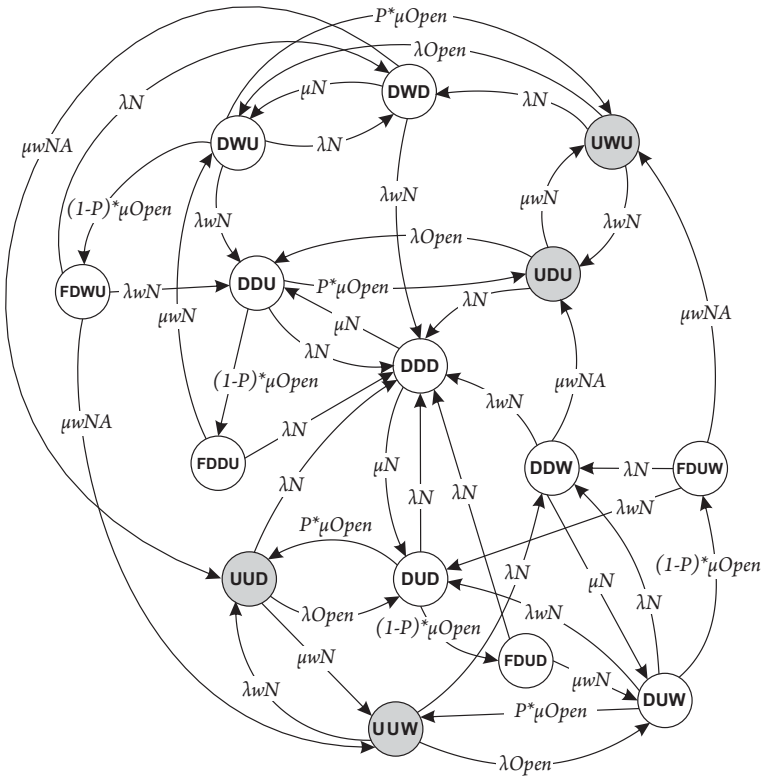


Figura 4.13: CTMC para arquitetura redundante nos Nós

O estado inicial da CTMC é o *UUW* onde o serviço está disponível, e a partir dele podem ser alcançado os estados *UUD*, quando acontece uma falha no Nó secundário em estado *warm standby*, o estado *DUW*, quando acontece uma falha no serviço, e acontecendo uma falha no Nó primário será direcionado ao estado *DDW*. Do estado *UUD* pode acontecer uma falha na plataforma de MBaaS e o sistema ser direcionado para o estado *DUD*, ainda do estado *UUD* o Nó primário pode falhar alcançando assim o estado onde todos os componentes estão indisponíveis *DDD*. A partir do estado *DUD* devido a política de manutenção supracitada, será dado prioridade ao reparo do serviço, assim existe uma probabilidade *P* de alcançar o estado *UUD*, caso esse processo de reparo falhe o sistema será direcionado para o estado *FDUD*, que representa a detecção de falha no processo de reparo da plataforma de MBaaS. A partir do estado *DUD*, também pode acorrer uma falha no Nó principal levando o sistema para o estado inativo *DDD*. Do *FDUD*, estado de detecção da falha no processo de reparo da plataforma de MBaaS, de acordo com a política de manutenção adotada, pode acontecer o reparo Nó secundário tornando este *warm standby* indo para o estado *DUW*, ou pode acontecer também uma falha no Nó principal direcionando para o estado *DDD*.

Do estado *DUW* de acordo com a política a primeira tentativa é realizar o reparo na plataforma de MBaaS, caso tenho sucesso, alcançará o estado inicial *UUW*, se acontecer uma falha nesse procedimento o sistema será direcionado para o estado *FDUW*, uma falha no Nó principal leva o sistema para o estado *DDW*. Do estado de detecção de falha *FDUW*, pode o

Nó em estado *warm standby* falhar resultando no estado *DUD*, deste estado de detecção pode acontecer o chaveamento do Nó em *warm standby* para Nó ativo, aqui é considerado o tempo de chaveamento mais o tempo de inicialização da VM mais o tempo de inicialização do serviço, este processo resultará no estado disponível *UWU*. A partir do estado *DDW* uma falha no Nó *warm standby* pode acontecer direcionando o sistema para o estado *DDD*, pode ocorrer também o chaveamento do Nó *warm standby* resultando no estado onde a plataforma encontra-se ativa *UDU*, outro evento que pode ocorrer é o reparo do Nó primário *DUW*.

No estado *UDU* o sistema está disponível, mesmo o Nó primário estando em falha. Deste estado podem ser alcançados os estados *UWU* que consiste no reparo do primeiro Nó para *warm standby*, o estado *DDU* que representa a falha na plataforma de MBaaS e pode ocorrer uma falha no segundo Nó indo para o estado totalmente inativo *DDD*. Do estado *UWU* onde o serviço também está ativo, pode acontecer uma falha na plataforma MBaaS e assim alcançar o estado *DWU*, pode acontecer também uma falha no Nó em *warm standby* direcionando o sistema para o estado *UDU*, como também pode acontecer uma falha no segundo Nó, indo o sistema para o estado *DWD*. Do estado *DWD* podem acontecer três eventos: primeiro uma falha no Nó *warm standby* levando o sistema ao estado *DDD*, segundo pode acontecer o reparo do segundo Nó *DWU* e por fim, pode acontecer o chaveamento onde o Nó em *warm standby* para ativo *UUD*, como já foi citado esse processo abrange o chaveamento somado ao tempo de instanciação da VM mais o tempo de inicialização do serviço.

No estado *DWU* o sistema está indisponível devido a uma falha na plataforma MBaaS. De *DWU* podem ser alcançados os estados *DWD* devido a uma falha no segundo Nó, *DDU* devido a uma falha no Nó em *warm standby*, *UWU* pode ser alcançado com o reparo da plataforma de MBaaS, caso esse processo de reparo falhe o estado *FDWU* que representa o estado de detecção de falha do processo de recuperação da plataforma pode ser alcançado. Do estado de detecção de falha *FDWU* pode ocorrer uma falha no Nó em *warm standby* e o sistema ser direcionado para o estado *DDU*, pode ocorrer também uma falha no segundo Nó sendo o sistema direcionado ao estado *DWD*, como também pode ocasionar o chaveamento do Nó em *warm standby* para ativo, levando o sistema ao estado *UUW*. Do estado *DDU* onde o sistema encontra-se indisponível devido uma falha na plataforma MBaaS, pode-se alcançar o estado *UDU* que devido a política de manutenção já descrita prioriza o reparo do serviço, este reparo pode falhar alcançando o estado de detecção de falha *FDDU*, pode acontecer uma falha no segundo Nó levando o sistema ao estado *DDD*.

Do estado *FDDU* onde é detectada a falha da recuperação do serviço no estado *DDU* pode ser alcançado o estado *DWU* devido ao reparo em *warm standby* do primeiro Nó, como pode ser alcançado o estado *DDD* devido a uma falha no segundo Nó. No estado *DDD* como já citado todos os componentes encontram-se em falha, porém deste estado pode ocorrer um reparo no Nó secundário *DDU*, pode também ocorrer uma reparo no Nó primário *DUD*. A nomenclatura de todos os estados está melhor descrita na Tabela 4.3.

A taxa de falha e reparo de ambos os Nós é representada por λ_N e μ_N respectivamente.

Tabela 4.3: Nomenclatura utilizada nos estados

Estados	Descrição
UUW	Plataforma MBaaS *up*, Nó 1 *up* e Nó 2 *warm*
UUD	Plataforma MBaaS *up*, Nó 1 *up* e Nó 2 *down*
UDU	Plataforma MBaaS *up*, Nó 1 *down* e Nó 2 *up*
UWU	Plataforma MBaaS *up*, Nó 1 *warm* e Nó 2 *up*
DUD	Plataforma MBaaS *down*, Nó 1 *up* e Nó 2 *down*
DUW	Plataforma MBaaS *down*, Nó 1 *up* e Nó 2 *warm*
DDW	Plataforma MBaaS *down*, Nó 1 *down* e Nó 2 *warm*
DWD	Plataforma MBaaS *down*, Nó 1 *warm* e Nó 2 *down*
DWU	Plataforma MBaaS *down*, Nó 1 *warm* e Nó 2 *up*
DDU	Plataforma MBaaS *down*, Nó 1 *down* e Nó 2 *up*
DDD	Plataforma MBaaS *down*, Nó 1 *down* e Nó 2 *down*
FDUD	Detecção de falha no reparo da Plataforma MBaaS, Nó 1 *up* e Nó 2 *down*
FDUW	Detecção de falha no reparo da Plataforma MBaaS, Nó 1 *up* e Nó 2 *warm*
FDWU	Detecção de falha no reparo da Plataforma MBaaS, Nó 1 *warm* e Nó 2 *up*
FDDU	Detecção de falha no reparo da Plataforma MBaaS, Nó 1 *down* e Nó 2 *up*

Quando um Nó encontra-se no estado *warm standby* a taxa de falha desse Nó é representada por λ_wN, já o reparo para um Nó retornar ao estado *warm standby* é representado por μ_wN. A taxa do chaveamento do Nó em *warm standby* para ativo é representado por μ_wNA. Para o reparo da plataforma de MBaaS é utilizado a taxa μ_Open multiplicado pela probabilidade de sucesso P, a taxa μ_Open foi adquirida através do inverso do tempo médio de reparo obtido na CTMC da Figura 4.1, bem como a taxa de falha λ_Open que foi obtida através do inverso do tempo médio de falha. Quando o reparo na plataforma de MBaaS não é possível ser executado, o sistema é direcionado para um estado de detecção de falha a uma taxa que corresponde a $(1-P)$ multiplicado pela taxa de reparo da plataforma (μ_Open).

Devido a quantidade de estados presente nesta CTMC, obtivemos uma fórmula muito extensa, a qual ficou inviável ser apresentada nesta dissertação. Contudo, resolvemos a CTMC da Figura 4.13 numericamente, para isso utilizamos as ferramentas *Symbolic Hierarchical Automated Reliability and Performance Evaluator* (SHARPE) (TRIVEDI; PULIAFITO, 1997) e Mercury (SILVA et al., 2013), (SILVA et al., 2015). Também é possível o cálculo da disponibilidade estacionária através da seguinte Expressão.

$$A_{Sub-sistemaMBaaS} = \pi_{UUW} + \pi_{UUD} + \pi_{UDU} + \pi_{UWU} \tag{4.26}$$

Onde $A_{Sub-sistemaMBaaS}$ corresponde a soma das probabilidades dos estados onde o serviço estará disponível.

A fórmula para o cálculo da disponibilidade no modelo de alto nível RBD pode ser representada pela Equação 4.27. Já indisponibilidade é definida na expressão da Equação 4.28,

enquanto o *downtime* é representado na Equação 4.29.

$$A_{nuvem\_no\_redundante} = A_{Frontend} \times A_{NAS} \times A_{Sub-sistemaMBaaS} \tag{4.27}$$

$$UA_{nuvem\_no\_redundante} = 1 - A_{nuvem\_no\_redundante} \tag{4.28}$$

$$Downtime_{nuvem\_no\_redundante} = UA_{nuvem\_no\_redundante} \times 8760h \tag{4.29}$$

4.6 Modelos para Arquitetura com Redundância no *Frontend* e no Nó

Nesta seção é apresentado o modelo com redundância *warm standby* no *Frontend* e no Nó, esta arquitetura é representada através da Figura 4.14, como nas outras arquiteturas apresentadas, o principal objetivo dessa proposta para a plataforma de MBaaS OpenMobster é aumentar a sua disponibilidade estacionária, podendo oferecer aos usuários finais um serviço mais confiável. O modelo que denota essa arquitetura é o modelo de alto nível RBD da Figura 4.15, que é composto por três submodelos: o submodelo redundante *Frontend* já foi refinado no modelo CTMC proposto por (DANTAS et al., 2012) da Figura 4.10, o submodelo NAS que é o dispositivo de armazenamento, como já foi descrito nas seção anteriores, e o submodelo redundante Sub-sistema MBaaS, que também já foi refinado no modelo CTMC da Figura 4.13. O modo operacional dessa arquitetura compreende que, para ficar disponível, pelo menos um dos *Frontends* deve estar ativo, assim como o Nó, no mínimo um deve estar ativo para que o serviço seja provido corretamente. Para o serviço ficar indisponível é necessário, um problema nos dois *Frontend* ou nos dois Nós. A Expressão Lógica 4.30 demonstra esse modo operacional.

$$Plataforma_{UP} = (F1_{UP} \vee F2_{UP}) \wedge (NO1_{UP} \vee NO2_{UP}) \tag{4.30}$$

A partir dessa arquitetura é também possível a obtenção de uma expressão algébrica para calcular a disponibilidade da infraestrutura da nuvem completa $A_{SistemaFN}$. Dessa forma, a Equação 4.31 calcula a disponibilidade de acordo com a regra de composição de componentes em série (KUO; MING, 2002). O valor de $A_{Frontend_reduntante}$ foi retirado da Equação 4.21 e o valor de $A_{Sub-sistemaMBaaS}$ foi retirado da Equação 4.26. A expressão algébrica que defini a indisponibilidade desse ambiente está representada na Equação 4.32, e o *downtime* na Equação 4.33.

$$A_{nuvem\_no\_front} = A_{Frontend\_reduntante} \times A_{Sub-sistemaMBaaS} \tag{4.31}$$

Onde:

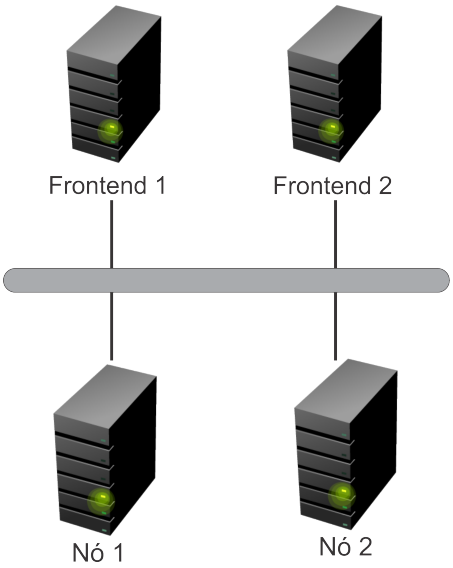


Figura 4.14: Arquitetura com redundância nos Nós e no *Frontend*

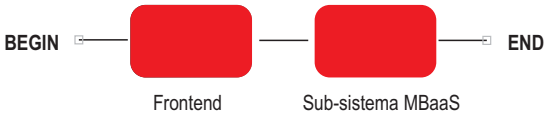


Figura 4.15: RBD com redundância no *Frontend* e nos Nós

$A_{nuvem_{no}front}$ representa a disponibilidade da nuvem com redundância no Nó e no *Frontend*
$A_{Frontend_{r}eduntante}$ representa a disponibilidade dos *Frontends* redundantes
$A_{Sub-sistemaMBaaS}$ representa a disponibilidade da Plataforma MBaaS

$$UA_{nuvem\_no\_front} = 1 - A_{nuvem\_no\_front} \tag{4.32}$$

$$Downtime_{nuvem\_no\_front} = UA_{nuvem\_no\_front} \times 8760h \tag{4.33}$$

4.7 Considerações Finais

Este capítulo apresentou um conjunto de modelos responsável por representar os comportamentos de falhas e reparos das arquiteturas propostas para o ambiente MBaaS, sob o ponto de vista da disponibilidade. Devido à complexidade destes comportamentos, os modelos foram concebidos através de modelagem hierárquica heterogênea. A princípio foi construído o modelo da arquitetura básica, a partir deste modelo foi proposto um modelo com um processo de recuperação automática na JVM. Como na modelagem da arquitetura básica, foi modelado o serviço através de uma CTMC, este modelo serviu de insumo para o modelo construído para arquitetura de nuvem sem redundância. Uma análise de sensibilidade foi aplicada no ambiente de nuvem sem redundância, guiado por essa análise foi proposta mais três arquiteturas: uma com

redundância no *Frontend*, outra com redundância no Nó, e uma com redundância em ambos, *Frontend* e Nó, o mecanismo de redundância considerado foi o *warm standby*. A fim de calcular a disponibilidade um conjunto de fórmulas fechadas foram obtidos.

O próximo capítulo apresenta o processo de validação do modelo da arquitetura básica, modelo este que serviu como base para os demais modelos.

5

Validação do Modelo da Arquitetura Básica

Para verificar se o comportamento do modelo da Figura 4.2 da Seção 4.1 do capítulo anterior é semelhante ao ambiente real, isto é, se este modelo não tem evidências para ser rejeitado como um modelo que representa o ambiente de execução, um experimento de injeção de falhas foi realizado em um ambiente de testes, onde ambos (modelo e ambiente de execução) foram submetidos às mesmas condições de entrada. Posteriormente a este passo, realizou-se a validação do modelo através dos cálculos estatísticos propostos por (KEESEE, 1965), que consistem em calcular o intervalo de confiança da disponibilidade. A Figura 5.1 apresenta um *workflow* do processo de validação, é possível verificar que podemos dividir esse processo em duas grandes áreas, são elas: Injeção de falhas e Validação, que são descritos em detalhes nas seguintes seções.

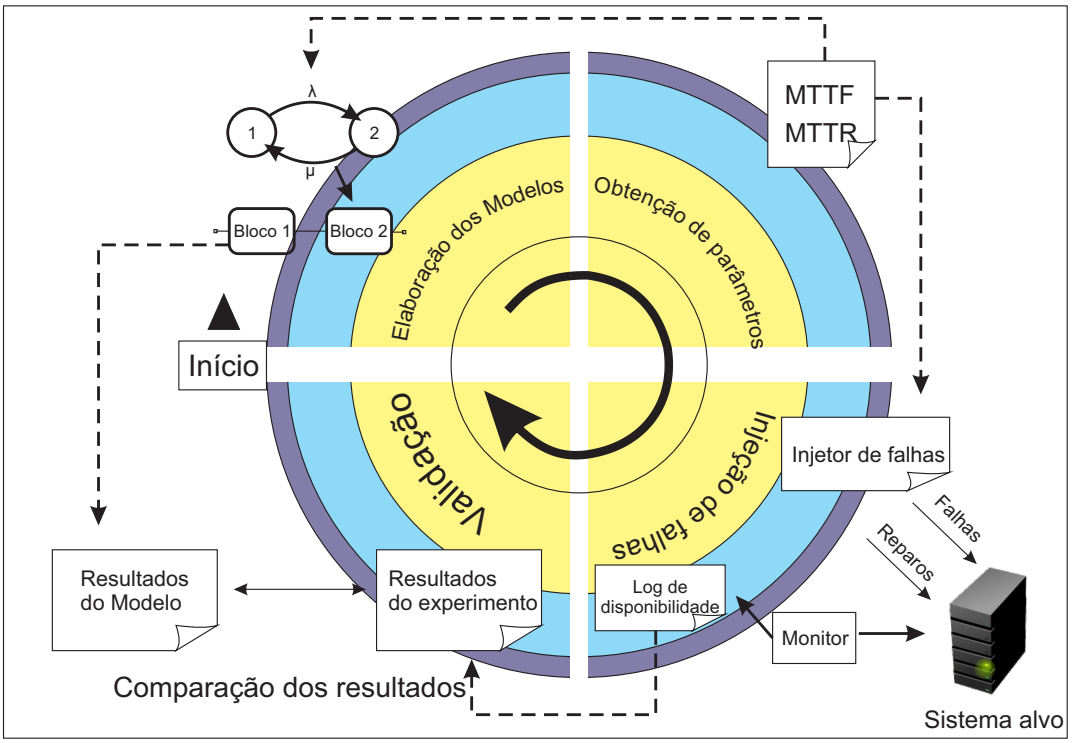


Figura 5.1: Workflow da validação

5.1 Injeção de Falha e Ambiente de Teste

Injeção de falhas é um método importante para avaliar experimentalmente medidas de dependabilidade em sistemas de computador (EJLALI et al., 2003). Simplificando, como já apresentado, injeção de falhas é uma técnica que consiste em inserir de maneira artificial falhas no sistema. De acordo com (VACARO; WEBER, 2006), essa abordagem visa à emulação de falhas de hardware, sem comprometer fisicamente o ambiente, e a análise do comportamento do sistema na exposição dessas falhas. Além das já citadas, podemos enumerar mais duas categorias principais de experimentos de injeção de falhas, são elas: injeção de falhas em modelos de sistemas simulados, e injeção de falhas em sistemas físicos (EJLALI et al., 2003). Esta pesquisa utilizou a técnica de injeção de falhas em sistemas físicos a nível de *software*, com gatilho baseado em *Time-out*. O objetivo foi verificar o comportamento da plataforma de MBaaS OpenMobster em um ambiente exposto a falhas.

Para a caracterização e análise do ambiente, garantimos que apenas os componentes, parte integrante da plataforma de MBaaS OpenMobster estariam em execução: o Banco de Dados, a JVM, o JBoss e a própria plataforma. Foi verificado também que o único serviço executando na porta 80 era a plataforma de MBaaS OpenMobster, essa verificação se torna importante por conta do monitoramento do serviço, que foi realizado utilizando a ferramenta [1], dessa maneira foi possível verificar quando o serviço estava ativo, podendo assim gerar um *log* da disponibilidade do sistema.

Para construção do ambiente foram utilizados dois computadores *desktops* com configurações diferentes, em um dos computadores foi instalado a plataforma de MBaaS OpenMobster e seus componentes e o outro foi utilizado como monitor e injetor de falhas, as configurações dos mesmos são: um com Processador Intel core i7, 4 GB de RAM, HD 500 GB SATA, o outro um Processador AMD Athlon Dual Core, 768 MB de RAM, HD 160 GB SATA, respectivamente, no primeiro computador foi instalado o serviço do OpenMobster, nesta máquina foram instalados também o $JBoss5.1.0GA$ e o $Java1.7.0_25$. Estes computadores estavam conectados através de uma rede *ethernet* utilizando cabeamento CAT 5e, através de um *switch* de 8 portas 10/100mbs, o sistema operacional utilizado em ambas as máquinas foi o Ubuntu 12.04. A Figura 5.2 ilustra o ambiente montando.

O ambiente foi supervisionado durante um período de 120 horas. Como o serviço da plataforma de MBaaS OpenMobster foi inicializado na porta 80, a ferramenta *nmap* foi utilizada para escutar esta porta. Algoritmos foram iniciados no segundo computador para injetar falhas em seis níveis do sistema: *Hardware*, Sistema Operacional, Banco de Dados, JVM, JBoss, e a plataforma de MBaaS OpenMobster. O comportamento do sistema foi monitorado através de um processo de injeção de falhas. Os algoritmos de injeção de falhas foram escritos com a linguagem *Shell script*, devido a extensão dos algoritmos, representamos nos Apêndices B

[1] *nmap* é um utilitário de código aberto, cujas funções incluem a descoberta de rede, administração, auditoria de segurança, bem como o escaneamento de portas (LYON; FYODOR, 2009).

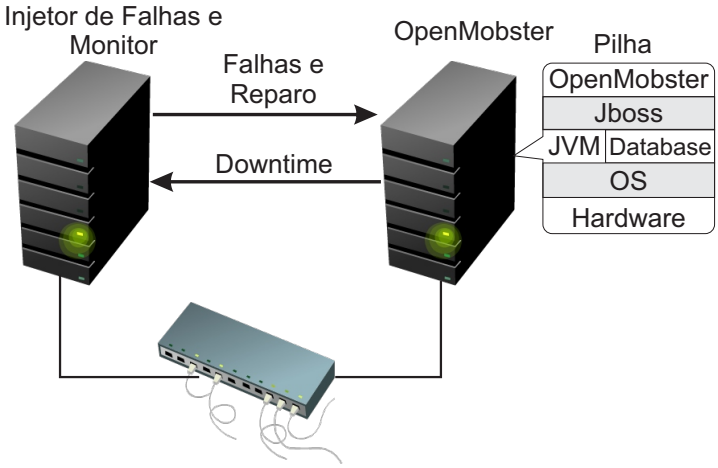


Figura 5.2: Componentes do ambiente de teste

e B apenas os algoritmos de injeção de falha para o Jboss, e para a JVM. Os seis algoritmos foram executados em paralelo e cada algoritmo tem a injeção de falhas direcionada para um componente específico do sistema, e um algoritmo é responsável por fazer o monitoramento do sistema. Este algoritmo descrito no Apêndice C gerou um *log* que foi utilizado para realizar o cálculo do intervalo de confiança proposto por (KEESEE, 1965).

Para permitir a execução paralela, a hierarquia dos componentes teve que ser respeitada, sendo assim foi implementada uma política de exclusão mútua nos componentes de mais baixo nível, por exemplo, quando uma falha de *hardware* for injetada, o algoritmo através de chamada ao sistema, interrompe a execução dos processos dos outros algoritmos, liberando-os quando o reparo for realizado.

A utilização de injeção de falhas para validação do modelo exige que as características dos tempos de falha e reparo sejam definidas previamente. Deste modo, a ferramenta R (PROJECT, 2014) foi adotada para criar os tempos de injeção de falha e reparo de acordo com uma distribuição exponencial, com base nos MTTF e MTTR dos componentes (ver Tabela 5.1), a obtenção desses tempos é detalhada no Capítulo 6, na Seção 6.1. Dado que os tempos de falha são geralmente muito longos, é necessário reduzi-los para viabilizar a validação. Propõe-se a redução utilizando um fator constante (SOUZA et al., 2013), desta forma, estes por sua vez foram reduzidos a fim de acelerar o experimento, foi adotado um fator de redução para os eventos de falha de 600, e de 100 para os eventos de reparo.

A Figura 5.3 exemplifica através de um diagrama de atividade o processo básico para injeção de falhas, onde é possível verificar que inicialmente o algoritmo gera um tempo exponencialmente distribuído para a injeção de falha. Após isso um comando de espera de acordo com o tempo gerado é executado, quando o tempo gerado é alcançado a injeção de falha é disparada, essa falha irá variar de acordo com o componente que o algoritmo é direcionado, por exemplo, se a falha for direcionada para o componente Jboss é executado um comando *"shutdown.sh"* no *host* remoto através de um cliente *Secure Shell* (SSH).

Tabela 5.1: Taxas utilizadas na injeção de falhas

Componente	**Taxa de falha** (λ) s^{-1}	**Taxa de reparo** (μ) s^{-1}
Hardware	0,0000190	0,0034722
OS	0,0000576	0,0069444
JVM	0,0002115	0,0138888
JBoss	0,0002115	0,0138888
OpenMobster	0,0002115	0,0138888
Database	0,0001157	0,0092592

Depois de injetada a falha, o algoritmo irá gerar o tempo exponencialmente distribuído para o reparo, o mesmo processo descrito anteriormente é feito, um comando de espera é disparado, e após o tempo ser alcançado será realizado um reparo, da mesma forma esse reparo irá variar de acordo com o componente a que o algoritmo está atrelado, depois do processo o algoritmo executa o *loop* e retorna para inicio fazendo todo o processo novamente, a condição de parada é estipulada pelo avaliador.

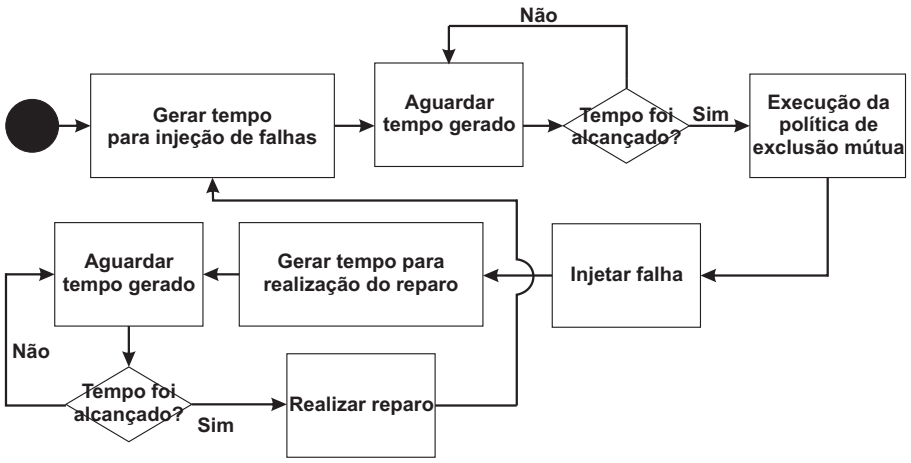


Figura 5.3: Diagrama de atividade da injeção de falha

5.2 Validação

O principal objetivo da validação é demonstrar que o modelo é uma representação razoável do sistema real e reproduz o comportamento do sistema com fidelidade suficiente para satisfazer os objetivos da análise. O objetivo da validação é quantificar a confiança na capacidade de previsão do modelo em comparação com dados experimentais (AERONAUTICS; STAF, 1998). De acordo com (SARGENT, 1998), a validação de modelo significa a comprovação de que um modelo computadorizado dentro do seu domínio de aplicação possui uma gama satisfatória de rigor compatível com a aplicação a que se destina o modelo. Este trabalho adotou

o método de validação proposto por (KEESEE, 1965), que envolve o cálculo do intervalo de confiança da disponibilidade.

O método de (KEESEE, 1965) propõem que a validação seja realizada através da comparação dos resultados obtidos com o modelo, para as medidas recolhidas diretamente a partir do ambiente de testes. Deste modo, a disponibilidade do sistema real foi encontrado a partir do *log* gerado pelo algoritmo monitor. No qual, encontramos o tempo total para as falhas S_n, que foi igual a 91977 segundos, e o tempo total de reparos Y_n que foi igual a 10505 segundos. Desta maneira, podemos encontrar o valor de $\hat{\rho}$, conforme a Equação 5.1.

$$\hat{\rho} = \frac{Y_n}{S_n} = \frac{10505}{91977} = 0,114213336 \tag{5.1}$$

Com o valor de $\hat{\rho}$ definido, analisamos a quantidade total de eventos injetados no ambiente, que correspondeu a 856 eventos de falha e reparo, este valor é utilizado como o grau de liberdade para o cálculo da distribuição F. A partir deste ponto foram calculados os valores mínimos e máximos da distribuição F com 95% de confiança, e encontrar os valores ρ_U e ρ_L, para a isso é utilizamos as Equações 5.2 e 5.3.

$$\rho u = \frac{\hat{\rho}}{f856;856;1-\frac{\alpha}{2}} = \frac{0,114213336}{0,8745} = 0,1306041 \tag{5.2}$$

$$\rho l = \frac{\hat{\rho}}{f856;856;\frac{\alpha}{2}} = \frac{0,114213336}{1,143} = 0,0999241 \tag{5.3}$$

onde:

α representa o nível de confiança

A partir dos valores de ρ_U e ρ_L definidos, de acordo com (KEESEE, 1965) podemos encontrar o intervalo de confiança da disponibilidade, ou seja, A_U e A_L, para isso utilizamos a seguinte Equação.

$$(\frac{1}{1+\rho u}, \frac{1}{1+\rho l}) = (\frac{1}{1+0,130604158}, \frac{1}{1+0,099924178}) = (0,8844828; 0,9091535) \tag{5.4}$$

Com o intervalo de confiança definido, a validação do modelo proposto na arquitetura básica foi realizada. Utilizando a Equação 4.2 do modelo RBD da Figura 4.2, é possível encontrar a disponibilidade de 0,8969447, a qual encontra-se dentro do intervalo de confiança definido na Tabela 5.2. Portanto, o modelo proposto não tem evidências para ser refutado como modelo que representa o comportamento do ambiente real, pois ambos estão consistentes entre si.

Tabela 5.2: Intervalo de confiança para A and ρ

Intervalo de confiança (95%)		
ρ	ρ_U	0,1306041
	ρ_L	0,0999241
A	A_U	0,8844828
	A_L	0,9091535

5.3 Considerações Finais

Neste capítulo, foi apresentado o processo de validação do modelo da arquitetura básica, que por sua vez foi utilizado como insumo para as arquiteturas subsequentes. O processo de validação empregado na presente pesquisa utilizou como método o cálculo do intervalo de confiança que foi detalhado anteriormente. Para tal, como descrito, construímos um *testbed* que teve como objetivo principal expor o sistema a uma quantidade razoável de falhas e reparos, e assim coletar informações que foram utilizadas como entrada para o cálculo do intervalo de confiança. Ao final do processo obtivemos como resultado um modelo que representa o ambiente real, e desta maneira, demonstra ser um modelo que tem credibilidade para ser utilizado na avaliação de disponibilidade do ambiente, bem como na proposição de novas arquiteturas.

6

Estudos de Caso

Este capítulo apresenta estudos de casos criados com o objetivo de avaliar o impacto das alternativas arquiteturais propostas para ambientes MBaaS, sob a ótica da disponibilidade. Essas arquiteturas foram apresentadas nos modelos do Capítulo 4. Avaliação de dependabilidade em ambientes que provêm algum tipo de serviço é uma atividade importante, uma vez que a obtenção de métricas como disponibilidade estacionária e indisponibilidade anual podem auxiliar os administradores desses ambientes na projeção de melhorias arquiteturais, bem como na definição de políticas de manutenção.

Desta maneira, os estudos de caso aqui propostos têm como base as métricas disponibilidade e *downtime*. Assim, primeiro estudo de caso apresentado na Seção 6.1, realiza uma comparação entre a disponibilidade e *downtime* da arquitetura básica com a arquitetura com processo de recuperação automática na JVM, bem como é analisado a eficiência do processo de recuperação automática. O segundo estudo de caso mostrado na Seção 6.2, traz uma análise do impacto dos componentes da arquitetura básica sobre a disponibilidade. O terceiro estudo de caso detalhado na Seção 6.3, apresenta a avaliação da disponibilidade da plataforma de MBaaS OpenMobster sobre uma infraestrutura de nuvem privada sem redundância. No quarto estudo de caso da Seção 6.4, é realizada uma avaliação de disponibilidade em um ambiente de nuvem privada com redundância no *Frontend*. O quinto estudo de caso da Seção 6.5, mostra a avaliação de disponibilidade da plataforma de MBaaS em relação a uma infraestrutura de nuvem privada com redundância nos Nós. Para investigar os efeitos do uso combinado de redundância no *Frontend* e nos Nós na disponibilidade, apresentamos um sexto estudo de caso na Seção 6.6.

6.1 Avaliação da Arquitetura Básica da Plataforma de MBaaS

A avaliação de disponibilidade é uma atividade importante em ambientes que precisem cumprir acordos de níveis de serviço (SLA). Assim, os modelos apresentados no Capítulo 4, na Seção 4.1 foram concebidos com o objetivo de avaliar a disponibilidade neste ambiente. Desta forma, este estudo de caso tem como objetivo principal avaliar a disponibilidade estacionária da plataforma de MBaaS na sua arquitetura básica, mostrando como o sistema pode se comportar

em condições normais, bem como é apresentada uma avaliação da disponibilidade quando é empregado um processo de recuperação automática em um dos componentes de mais baixo nível da plataforma, que é a JVM. Ademais, este estudo pode fornecer a administradores uma visão de como é o comportamento da plataforma sob a ótica da disponibilidade com a utilização de um processo de recuperação automática em um dos seus componentes.

Para realizar a avaliação foram inseridos valores nos parâmetros apresentados nos modelos CTMC das Figuras 4.1 e 4.3 do Capítulo 4. Esses parâmetros e valores são descritos da seguinte maneira: os parâmetros μ_O, μ_B e μ_J possuem valores iguais e foram obtidos em (SOUZA et al., 2013), os parâmetros de falha λ_O, λ_B e λ_J também possuem valores iguais e forma retirados de (DANTAS et al., 2012). Tais valores foram escolhidos porque correspondem a falha e reparo de um sistema semelhante ao avaliado neste trabalho. Os parâmetros μ_D e λ_D possuem valores diferentes, e foram adaptados de (TEOREY; NG, 1998), que realiza uma análise de confiabilidade e desempenho na base de dados centralizada ou distribuída. A taxa de reparo para o processo de recuperação automática é representada pelo parâmetro $f\mu\_J$ e foi definido como um valor aproximado para uma recuperação automática, já a probabilidade de sucesso do processo de recuperação automática F foi retirado de (BAUER E.; EUSTACE, 2011). Todos esses valores estão listados na Tabela 6.1.

Tabela 6.1: Valores dos parâmetros para os modelos CTMCs

Parâmetro	Descrição	Taxa (1/h)
$\lambda\_O = \lambda\_B = \lambda\_J$	Taxa de falha	0,001269
$\mu\_O = \mu\_B = \mu\_J$	Taxa de reparo	0,5
λ_D	Taxa de falha Banco de Dados	0,000694
μ_D	Taxa de reparo Banco de Dados	0,333333
$f\mu\_J$	Taxa de reparo (processo de recuperação)	10
F	Probabilidade de sucesso do proc. de rec.	99%

Para a análise dos modelos hierárquicos baseados em RBD utilizou-se a ferramenta Mercury (SILVA et al., 2013), (SILVA et al., 2015), e para a avaliação das CTMCs utilizou-se a ferramenta Mercury (SILVA et al., 2013), (SILVA et al., 2015) e SHARPE (TRIVEDI; PULIAFITO, 1997). A fim de acelerar o experimento de injeção de falha, foi utilizado um fator de redução sobre os valores dos parâmetros apresentados na Tabela 6.1. Para a falha foi utilizado um fator de redução de 600, para o reparo foi empregado um fator de 100 e para o processo de recuperação automática um fator de 20, logo após esses valores foram transformados em segundos. Para manter a coerência, esses mesmos valores foram utilizados nas taxas dos modelos CTMC. Destes, foi possível extrair os seguintes resultados: disponibilidade; tempo médio de falha (MTTF), e tempo médio de reparo (MTTR). Estes valores foram utilizados nos modelos RBD da Figura 4.2. A Tabela 6.2 demonstra os resultados gerados pelas CTMCs em segundos. Os parâmetros utilizados no RBD (ver Figura 4.2) são apresentados na Tabela 6.3.

Tabela 6.2: Resultados dos modelos CTMCs

Descrição	Disponibilidade
CTMC arquitetura básica	0,909341
CTMC com proc. de rec. automática (99%)	0,919549

Tabela 6.3: Parâmetros para o modelo RBD

Componente	MTTF (s)	MTTR (s)
Hardware	52560	288
SO	17358	144
Arquitetura básica	1332,87392	132,883813
Arquitetura com processo de recuperação	1332,87392	116,612443

Os resultados gerados pelos modelos demonstram uma melhoria na disponibilidade do sistema quando o processo de recuperação automática é empregado no componente JVM. De acordo com os resultados apresentados na Tabela 6.4, pode-se afirmar que o uso do processo de recuperação automática reduz o tempo de indisponibilidade anual do sistema em 10%, e, portanto, verifica-se que a técnica de recuperação automática é algo importante a ser abordado no sistema aqui avaliado.

Tabela 6.4: Resultados da comparação entre os modelos

Metricas	Sistema com rec. automática	Sistema na arquitetura básica
Disponibilidade	0,907013	0,896944
Disponibilidade (9's)	1,0315800	0,986929
Downtime (h)	814,5626	902,7644

Para confirmar esta conclusão, fizemos uma variação da probabilidade de sucesso desse mecanismo. A variação foi de 0,8 a 0,99, variando de 0,01, sendo assim obtivemos vinte cenários. O aumento da disponibilidade do sistema em relação ao aumento da probabilidade de sucesso do processo de recuperação automática pode ser observado no gráfico da Figura 6.1, podendo-se verificar que quanto mais eficiente for o processo de recuperação automática, maior será a disponibilidade apresentada pelo sistema.

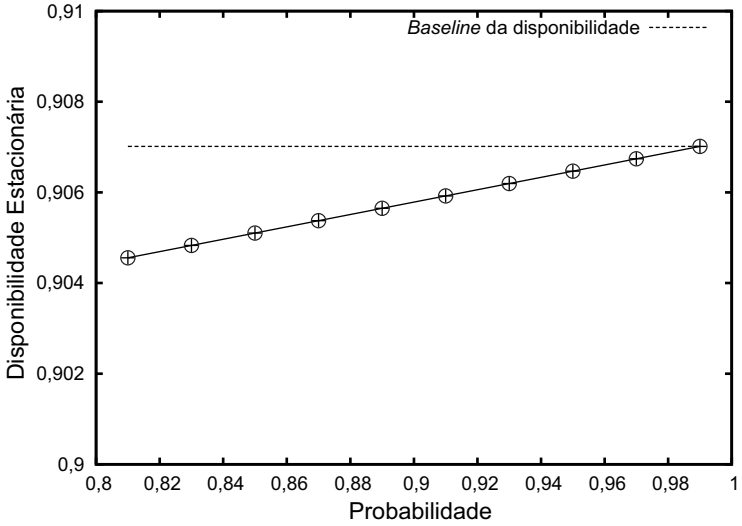


Figura 6.1: Probabilidade de sucesso do processo de recuperação automática

6.2 Avaliação do Impacto dos Componentes da Arquitetura Básica sobre a Disponibilidade

Ampliando a investigação dos modelos e com o objetivo em determinar qual componente da arquitetura básica é mais crítico, foi realizada uma análise de sensibilidade paramétrica, que como já fora mencionado tem o objetivo de determinar quais os fatores que são mais relevantes sobre as medidas ou saída de um modelo.

A Tabela 6.5 mostra a sensibilidade em escala da disponibilidade de todos os parâmetros do sistema. Os resultados são ordenados de acordo com os valores absolutos dos índices de sensibilidade. O valor absoluto reflete o quão fortemente um parâmetro pode influenciar na disponibilidade. Os valores negativos indicam que existe uma relação inversa entre os parâmetros e a disponibilidade do sistema.

Tabela 6.5: Ranking de sensibilidade dos componentes

Parâmetros	**Valores**
MTTRSystem	-0,0804508727
MTTFSystem	0,0804508727
MTTFOS	0,0082276311
MTTROS	-0,0082276311
MTTRHardware	-0,0054495912
MTTFHardware	0,0054495912

Os valores para $S^*_\theta(A)$ foram calculados com a ferramenta Mercury (SILVA et al., 2013), (SILVA et al., 2015), onde A é a disponibilidade do sistema de estado estacionário e θ é cada um

dos parâmetros do sistema (as taxas de falha e de reparo de cada componente). Os indicadores de sensibilidade também podem ser calculado com a seguinte Equação 6.1:

$$\begin{aligned} S_\theta(A) = & \frac{\partial A_{HW}}{\partial \theta} \times A_{SO} \times A_{SM} + \\ & A_{HW} \times \frac{\partial A_{SO}}{\partial \theta} \times A_{SM} + \\ & A\mathrm{HW} \times A_{SO} \times \frac{\partial A_{SM}}{\partial \theta} \end{aligned} \tag{6.1}$$

Onde:
A_{HW} representa a disponibilidade do *Hardware*
A_{SO} representa a disponibilidade do *Sistema Operacional*
A_{SM} representa a disponibilidade do *Sistema MBaaS*

Com as taxas de falha e de reparo do HW definidas como λ_{HW} e μ_{HW}, respectivamente, as expressões derivadas correspondentes são apresentadas nas Equações 6.2 e 6.3. As expressões derivadas para AOS e ASys são semelhantes aos de AHW, uma vez que esses subsistemas são representados através de RBDs.

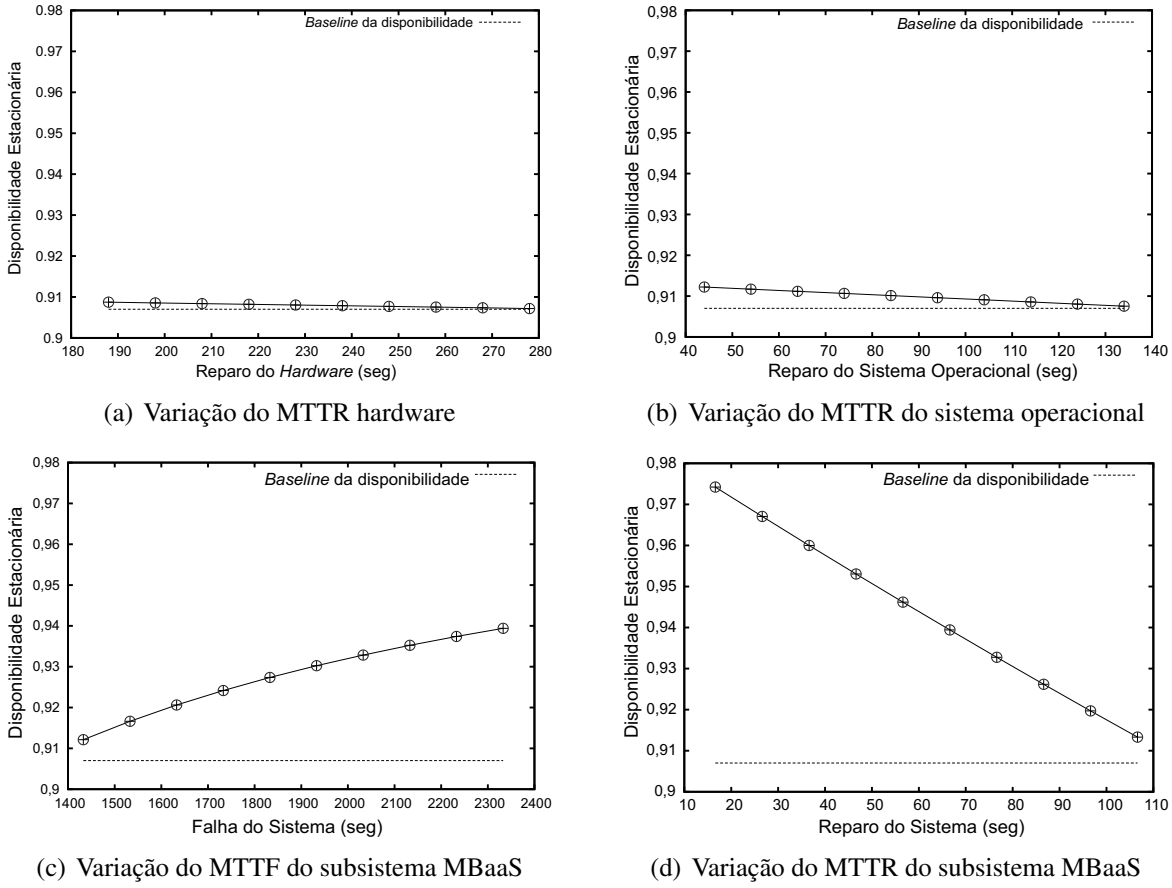
$$\frac{\partial A\mathrm{HW}}{\partial \lambda_{HW}} = -\frac{\mu_{HW}}{(\mu_{HW} + \lambda_{HW})^2} \tag{6.2}$$

$$\frac{\partial A\mathrm{HW}}{\partial \mu_{HW}} = -\frac{\mu_{HW}}{(\lambda_{HW} + \mu_{HW})^2} + \frac{1}{\lambda_{HW} + \mu_{HW}} \tag{6.3}$$

Os parâmetros *MTTRSystem* e *MTTFSystem* têm os valores mais altos de sensibilidade, por isso, pode-se afirmar que o componente Sub-sistema MBaaS tem o maior impacto sobre a disponibilidade de estado estacionário do sistema completo, e que qualquer alteração em seus parâmetros terá um grande impacto positivo ou negativo sobre a disponibilidade do sistema. Com uma análise mais detalhada é possível observar o efeito da disponibilidade ao variar os valores dos parâmetros na Figura 6.2. Note-se que os valores utilizados para análise foram os valores reduzidos e os mesmos estão convertidos em segundos.

No parâmetro *MTTFHardware*, onde variamos em um intervalo de 52560 a 53460 segundos, com uma variação de 100 em 100 segundos, podemos observar com essa variação um pequeno impacto na disponibilidade, pois a disponibilidade inicia em 0,907013 e finaliza em 0,907096, ou seja, uma redução de 0.09% o que podemos considerar como uma redução irrelevante, a *baseline* indica o impacto da disponibilidade com o MTTF real, confirmando o resultado da Tabela 6.5 que apresenta esse parâmetro como o de menor impacto na disponibilidade.

A Figura 6.2-a representa o gráfico referente ao parâmetro *MTTRHardware*, onde a variação foi de 10 em 10 segundos no intervalo de 188 a 288 segundos. Como já supracitado esse parâmetro tem um valor sensitivo negativo, pois com a redução do tempo de reparo é possível



(a) Variação do MTTR hardware

(b) Variação do MTTR do sistema operacional

(c) Variação do MTTF do subsistema MBaaS

(d) Variação do MTTR do subsistema MBaaS

Figura 6.2: Análise de sensibilidade de alguns parâmetros

verificar um pequeno aumento na disponibilidade em relação a *baseline*. A disponibilidade inicia em 0,9087330 com o menor tempo de reparo, e finaliza em 0,9070134 com maior tempo de reparo, com uma redução de 1,85% no *downtime*, e mesmo sendo um pouco maior que a redução em relação ao parâmetro *MTTFHardware*, continua sendo ainda uma redução muito pequena.

A Figura 6.2-b representa o parâmetro *MTTROS*, parâmetro este que está relacionado ao componente Sistema Operacional. Na Tabela 6.5, podemos observar que tanto esse parâmetro como o *MTTFOS* são o "meio termo", ou seja, não são os de maior impacto na disponibilidade e nem são os de menor impacto. Os valores do parâmetro *MTTFOS* foram variados de 17400 a 18400 segundos e o *MTTROS* de 40 a 140 segundos, variando de 100 em 100 segundos e 10 em 10 segundos respectivamente. No parâmetro *MTTFOS* a disponibilidade inicia em 0,9070134 e finaliza em 0,9074584, assim é possível verificar uma redução de 0,48% no *downtime*. Comparando com o parâmetro (*MTTFHardware*) este tem um maior impacto na disponibilidade, porém essa redução ainda é irrelevante. No parâmetro *MTTROS*, uma vez que tem um valor de sensibilidade negativo (como todos os parâmetros de reparo), aumentando os limites diminui a disponibilidade. Assim, a disponibilidade inicia em 0,9127501 e finaliza em 0,9070134, ou seja, uma redução de 6,17% no *downtime*.

As Figuras 6.2-c e 6.2-d representam os parâmetros *MTTFSystem* e *MTTRSystem* que estão relacionado à plataforma de MBaaS OpenMobster, como já foi constatado na Tabela 6.5

esse componente é o que mais impacta na disponibilidade de estado estacionário do sistema, assim fizemos a variação de 100 em 100 segundos no MTTF em um intervalo de 1332,87392 a 2332,87392 segundos, a disponibilidade inicia em 0,9070134 e finaliza com 0,9394097, ou seja, uma redução de 35% no *downtime*, então podemos observar uma melhora considerável nesse parâmetro, quanto ao MTTR, também foi feita uma variação de 10 em 10 segundos em um intervalo de 16,61244 a 116,6124 segundos, observando que com a redução desse parâmetro existe uma melhora considerável da disponibilidade, onde de acordo com a *baseline* a mesma está em 0,9070134 e com a redução vai para 0,9742252, ou seja, uma redução de 72% no *downtime*.

O resultado da análise dos gráficos indica que uma maior atenção deve ser dada para o componente sub-sistema MBaaS, porém não descartando os outros componentes. Desta forma, a aplicação de componentes redundantes pode trazer benefícios para esses ambientes, a proposição de uma arquitetura de nuvem privada para a implantação da plataforma de MBaaS, pode representar um ganho significativo em relação à níveis de disponibilidade. Estas propostas arquiteturais são avaliadas nos seguintes estudos de caso.

6.3 Avaliação da Disponibilidade na Arquitetura de Nuvem sem Redundância

Podemos afirmar que estudos de disponibilidade na infraestrutura básica da nuvem são particularmente úteis por diversas questões. Nesta seção avaliamos a disponibilidade da plataforma de MBaaS em uma arquitetura de nuvem privada, a qual é proposta na Seção 3.3 do Capítulo 3. Os modelos que correspondem a essa arquitetura estão ilustrados nas Figuras 4.4 e 4.7 do Capítulo 4.

Avaliação de disponibilidade nestes ambientes podem servir como referência para estudar o efeito de alterações na infraestrutura, como a aplicação de redundância em um componente, aplicação de um processo de recuperação automática, bem como fornecer um panorama do comportamento do sistema sob condições normais. Sendo assim, o objetivo deste estudo de caso é verificar o comportamento da disponibilidade estacionária da plataforma quando empregada em um ambiente de nuvem privada. Também é avaliado a eficiência do processo de recuperação automática na inicialização da plataforma na VM, onde é feito um comparativo do ambiente com e sem esse processo.

O RBD da Figura 4.4 é composto por sub-modelos, como o *Frontend* (Figura 4.5), o Nó (Figura 4.6), NAS e o Sub-sistema MBaaS. O submodelo *Frontend* é composto por subcomponentes, assim como o submodelo Nó, portanto, é necessário antes de apresentar os valores de *MTTF* e *MTTR* desses sub-modelos, apresentar os valores dos seus respectivos subcomponentes. O NAS é um componente único e seu tempo foi adotado de (TEOREY; NG, 1998). Na Tabela 6.6 apresentamos os valores inseridos nos parâmetros do modelo RBD relativo ao *Frontend* e ao Nó. É importante salientar que os tempos relativos ao *Hardware* e SO são os

mesmos para ambos os sub-modelos. Esses valores foram obtidos em (DANTAS et al., 2012) e (KIM; MACHIDA; TRIVEDI, 2009b).

Tabela 6.6: Parâmetros para os modelos RBDs *Frontend* e Nó

Bloco	Componente	MTTF (h)	MTTR (h)
Frontend	*Hardware*	8760	1,67
	SO	2895	1
	CLC	788,4	1
	CC	2788,4	1
	SC	2788,4	1
	Walrus	2788,4	1
Nó	KVM	2990	1
	NC	788,4	1

Como já explicado na Seção 4.2 do capítulo anterior, o bloco Sub-sistema MBaaS foi refinado com uma CTMC, que representa o serviço executando sobre uma *VM*, dessa forma os valores para os parâmetros desse bloco foram extraídos deste modelo. Os parâmetros utilizados na CTMC da Figura 4.7 estão descritos na Tabela 6.7. As taxas de falha e reparo da plataforma de MBaaS foram obtidas através da CTMC da Figura 4.1, que foi validada na Seção 5.2, já a taxa de reparo automática do serviço foi estipulada por aproximação com o tempo de inicialização do serviço somado ao tempo de detecção da falha, o tempo de inicialização do serviço é melhor explicado nos próximos parágrafos, e para o tempo de detecção assumimos que um *script* de monitoramento verifique o sistema a cada 30 segundos ($\cong$ 0,0083 h) e ao detectar a inatividade inicialize o processo de recuperação(ver Pseudo Código 1). A probabilidade de sucesso do processo de recuperação foi retirado de (BAUER E.; EUSTACE, 2011).

Tabela 6.7: Parâmetros de entrada do serviço sobre a nuvem - CTMC

Parâmetro	Descrição	Taxa (1/h)
λ_Open	Taxa de falha da plataforma	0,004501552
μ_Open	Taxa de reparo da plataforma	0,270914607
μVM	Taxa de instanciação da VM	98,039215686
λ_VM	Taxa de falha da VM	0,000347222
μ_OpenM	Taxa de reparo automática	74,626865671
P	Probabilidade de sucesso do processo de rec.	99%

O tempo de instanciação da VM equivale a soma do tempo de instanciação da mesma

Algorithm 1 Pseudo Código monitoramento do serviço

```
while true do
    sleep 30
    status ← verifyService

    while status == false do
        run −c openmobster −b "IPaddress"

        status ← verifyService

        if status == false then
            shutdown.sh −S
        end if
        end while
    end while
```

mais o tempo de inicialização do serviço, pois assumimos que o serviço encontra-se configurado para inicializar junto com o processo de inicialização do sistema operacional da VM. Portanto, a taxa μVM pode ser descrita na Expressão 6.5. O tempo de instanciação da VM pode ser descrito através da Equação 6.4, onde T_{inst} representa o tempo de instanciação da VM, tempo este que retiramos de (CAMPOS et al., 2015), no qual consideramos uma VM mais simples do tipo *m1.large*, com CPU de 2 *cores*, disco de 10 GB e memória de 512 de RAM. O tempo de instanciação para essa VM corresponde a 0,0051 h. T_{ini} corresponde ao tempo de inicialização do serviço.

$$T_{VM} = T_{inst} + T_{ini}; \tag{6.4}$$

$$\mu VM = \frac{1}{T_{VM}} \tag{6.5}$$

Para encontrar o tempo de inicialização do serviço foi realizado um experimento no ambiente de execução, no qual foi desenvolvido um *Script* com a linguagem *Shell Script*, para monitorar, inicializar e desativar o serviço em intervalos de 50 segundos, o monitoramento foi realizado a partir do *log* de inicialização do Jboss, este *Script* é melhor descrito no Apêndice D. Realizamos um total de 253 observações, os tempos foram coletados em segundos, a partir destes tempos encontramos o intervalo de confiança com 95% de confiança (ver Tabela 6.8), desta forma coletamos o valor 18.288 segundos, que corresponde a mediana deste intervalo.

Com os valores da CTMC definidos encontramos todos os valores dos parâmetros do RBD da Figura 4.4, como mostrado na Tabela 6.13, possibilitando assim extrair as métricas de dependabilidade dessa arquitetura.

Na Tabela 6.10 exibimos os resultados relativos à arquitetura de nuvem sem redundância, sendo possível observar uma disponibilidade de 0,9904 e um *downtime* de apenas três dias e

Tabela 6.8: Intervalo de confiança do tempo de inicialização do serviço

Intervalo de confiança (95%)		
T_{ini}	*mnimo*	17,610 s
	mximo	20,482 s

Tabela 6.9: Parâmetros de entrada do RBD da nuvem sem redundância

Componente	MTTF (h)	MTTR (h)
Frontend	180,72	0,96999
Nó	481,83	0,91000
NAS	1440	3
Sub-sistema MBaaS	76,0337	0,7275

meio. Para verificar o impacto do processo de recuperação automática do serviço, avaliamos a CTMC da Figura 4.7 sem o processo de recuperação automática, deste modo, retiramos da CTMC apenas o estado *FDU*. A Tabela 6.11 apresenta os resultados para este cenário, então, observamos que o processo de recuperação automática do serviço exerce grande impacto na disponibilidade desse ambiente, pois sem ele obtivemos uma disponibilidade de apenas 0,9745, enquanto o *downtime* aumentou cerca de 167%. Podemos observar essa comparação também através do gráfico da Figura 6.3. Os resultados demonstram que a proposta arquitetural da nuvem sem redundância e com o processo de recuperação automática no serviço pode aumentar significativamente a disponibilidade em ambientes MBaaS, dando assim subsídios e alternativas para os administradores desses ambientes. Outras propostas arquiteturais baseadas nesse ambiente são avaliadas nas próximas seções.

Tabela 6.10: Resultados do modelo RBD

Métricas	Valores
Disponibilidade	0,990494
Disponibilidade (9's)	2,022022
Downtime (h)	83,2725

Tabela 6.11: Resultados do modelo RBD sem o processo de recuperação automática

Métricas	Valores
Disponibilidade	0,974546
Disponibilidade (9's)	1,594249
Downtime (h)	222,97704

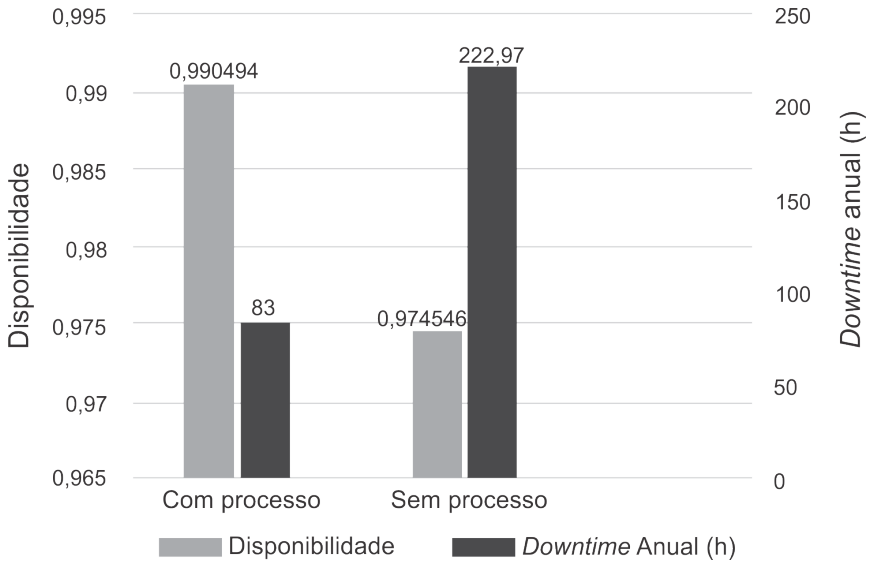


Figura 6.3: Comparação entre os modelos com e sem processo de recuperação automática

6.4 Avaliação da Disponibilidade na Arquitetura de Nuvem com Redundância no *Frontend*

A análise de sensibilidade é particularmente útil quando o administrador precisa saber a importância de um componente em relação à disponibilidade do sistema. A partir de uma análise de sensibilidade aplicada ao ambiente de nuvem sem redundância detalhada no Capítulo 4, foi verificado que o *Frontend* é o componente com maior criticidade na infraestrutura com plataforma MBaaS.

Isto posto, este estudo de caso tem como objetivo avaliar o impacto dessa arquitetura sobre a disponibilidade estacionária da plataforma de MBaaS. A arquitetura mencionada foi detalhada no Capítulo 4. A redundância no modo *warm standby* no *Frontend* de nuvem privada foi abordada por (DANTAS et al., 2012), dessa forma, utilizando a arquitetura de nuvem privada aplicamos a redundância no *Frontend*. Inicialmente, apresentamos os valores dos parâmetros de entrada que compõem os modelos analíticos apresentados nesse cenário, logo após é realizada a avaliação de disponibilidade com o intuito de verificar melhorias nesta métrica em relação a

arquitetura básica de nuvem sem redundância.

Os modelos analíticos que representam este ambiente foram detalhados no Capítulo 4, na Seção 4.4, o RBD que representa essa arquitetura é o ilustrado na Figura 4.4, os parâmetros utilizados no modelo são os mesmos apresentados na Tabela 6.13, com o exceção do *Frontend*, que nessa arquitetura foi modelado com a CTMC da Figura 4.10 proposta por (DANTAS et al., 2012). A Tabela 6.12 apresenta os valores dos parâmetros utilizado na CTMC. Esses valores foram obtidos em (DANTAS et al., 2012).

Tabela 6.12: Parâmetros de entrada para os *Frontend*s redundantes - CTMC

Parâmetro	Descrição	Taxa (1/h)
$\lambda\_s1 = \lambda\_s2$	Tempo Médio de Falha do *host*	0,005533
λ_i_s2	Tempo Médio para Falha do host inativo	0,004611
$\mu\_s1 = \mu\_s2$	Tempo Médio de Reparo do *host*	1,030927
sa_s2	Tempo Médio para Ativar o Sistema	200

A Tabela 6.13 apresenta os valores utilizados nos parâmetros do RBD da Figura 4.4, como supracitado os valores referentes ao MTTF e MTTR do bloco *Frontend* foram extraídos da CTMC 4.10.

Tabela 6.13: Parâmetros de entrada do RBD da nuvem com redundância no Front

Componente	MTTF (h)	MTTR (h)
Frontend	180,72	0,014373
Nó	481,83	0,91000
Sub-sistema MBaaS	76,0337	0,7275

Os resultados da Tabela 6.14, demonstra que a utilização de um *Frontend* redundante causa um impacto positivo na disponibilidade em relação a arquitetura de nuvem sem redundância, como também podemos observar uma redução de 71% no *downtime* em relação a arquitetura com redundância nos Nós. Estes valores reforçam o resultado da análise de sensibilidade apresentada no Capítulo 4, no qual é afirmado que o componente *Frontend* é o componente mais crítico nessa arquitetura.

Tabela 6.14: Resultados do modelo RBD

Métricas	Valores
Disponibilidade	0,997806
Disponibilidade (9's)	2,658769
Downtime (h)	19,21944

6.5 Avaliação da Disponibilidade na Arquitetura de Nuvem com Redundância no Nó

Nesta seção, avaliamos a disponibilidade da arquitetura de nuvem com redundância nos Nós, uma vez que esse componente foi apresentado na análise de sensibilidade da arquitetura de nuvem sem redundância como o terceiro mais crítico. Essa arquitetura foi descrita na Figura 4.11 do Capítulo 4. O principal objetivo deste estudo de caso é verificar o impacto causado na disponibilidade com a inclusão de um Nó redundante na arquitetura de nuvem descrita na Seção 4.2, os modelos analíticos que representam essa arquitetura estão presentes nas Figuras 4.12 a 4.13 do Capítulo 4.

Nessa arquitetura foi proposto uma mecanismo de redundância *warm standby* como descrito no Capítulo 4, este mecanismo pode ser implementado através do *software* de replicação *Heartbeat* (HA PROJECT, 2014), que de acordo com (DANTAS, 2013) é um software de gerenciamento de *cluster* que permite, em uma infraestrutura de *cluster*, identificar os *hosts*, ativos e inativos, por meio de troca de mensagens periódicas. A Tabela 6.15 apresenta os valores dos parâmetros utilizados nos componentes do modelo RBD da Figura 4.12. O sub-modelo *Frontend*, já teve os parâmetros de seus subcomponentes apresentados na Tabela 6.6. Como já citado, esses valores foram obtidos em (DANTAS et al., 2012) e (KIM; MACHIDA; TRIVEDI, 2009b), os valores referente ao NAS em (TEOREY; NG, 1998) e os valores do bloco Sub-sistema MBaaS foram obtidos a partir da CTMC da Figura 4.13.

Tabela 6.15: Parâmetros de entrada do RBD da nuvem com Nós redundantes

Componente	MTTF (h)	MTTR (h)
Frontend	180,72	0,969
NAS	1440	3
Sub-sistema MBaaS	144,421	0,040

Como mencionado, o bloco Sub-sistema MBaaS foi refinado com uma CTMC, na qual os parâmetros utilizados por esse modelo estão apresentados na Tabela 6.16. As taxas λN, μN, λwN e μwN foram retiradas de (KIM; MACHIDA; TRIVEDI, 2009b). De acordo com

(DANTAS et al., 2012), estando um Nó em um estado *warm standby* seu tempo médio entre as falhas é 20% maior do que a taxa de um Nó ativo, assim esse valor foi assumido. As taxas $\lambda Open$ e $\mu Open$ foram retiradas da CTMC da Figura 4.7 e corresponde ao serviço rodando sobre uma VM. Como a CTMC da Figura 4.13 tem uma política de priorização do serviço definida, P corresponde a probabilidade de sucesso para ativação do serviço. A taxa μwNA corresponde ao inverso do tempo de chaveamento e ativação do serviço no Nó em *warm standby* e para encontrar esse tempo utilizamos a Equação 6.6, na qual T_{chav} corresponde ao tempo de chaveamento, este valor foi retirado dos arquivos de configuração do *Heartbeat* e consideramos o valor de 0,005 h. T_{VM} é o tempo encontrado a partir da Equação 6.4, que como já explicado corresponde ao tempo de instanciação da VM somado ao tempo de inicialização do serviço.

$$\mu wNA = \frac{1}{T_{chav} + T_{VM}} \tag{6.6}$$

Tabela 6.16: Parâmetros de entrada do serviço com redundância nos Nós - CTMC

Parâmetros	Descrição	Taxa (1/h)
λ_Open	Taxa de falha da plataforma	0,004848
μ_Open	Taxa de reparo da plataforma	21,079702
μ_N	Taxa de reparo do Nó	1,098901
λ_N	Taxa de falha do Nó	0,002075
μ_wN	Taxa de reparo para o Nó em *warm standby*	30,030030
λ_wN	Taxa de falha do Nó em *warm standby*	0,001729
μwNA	Taxa de chav. e ativação para o Nó em *warm*	65,789473
P	Probabilidade de sucesso de ativar o serviço	99%

A Tabela 6.17 apresenta os resultados de dependabilidade referentes a essa arquitetura, onde podemos observar uma disponibilidade de 0,992313, que corresponde a um *downtime* de 67,33812 h, em comparação com o estudo de caso da arquitetura de nuvem sem redundância observamos uma redução de 19% no *downtime*, dessa forma essa infraestrutura se mostra viável do ponto de vista da disponibilidade do ambiente no lado servidor.

Tabela 6.17: Resultados do modelo RBD

Métricas	Valores
Disponibilidade	0,992313
Disponibilidade (9's)	2,114288
Downtime (h)	67,33812

6.6 Avaliação da Disponibilidade na Arquitetura de Nuvem com Redundância no *Frontend* e no Nó

A fim de apresentar uma infraestrutura de alta disponibilidade para ambiente de MBaaS, propomos a aplicação de uma redundância que chamamos de completa, pois é aplicada tanto no *Frontend* quanto no Nó, os dois cenários onde o ambiente MBaaS alcançou maiores disponibilidades. Os modelos correspondentes a esta arquitetura foram ilustrados no Capítulo 4. Como os modelos consideram a infraestrutura de nuvem privada com mecanismo de redundância *warm standby* no *Frontend* e no Nó, a análise de disponibilidade realizada pode servir como uma referência para estudar efeitos de alterações no ambiente, como por exemplo acréscimo de hardware.

Esta seção avalia a disponibilidade utilizando a arquitetura apresentada na Figura 4.14, a qual é representada pelos modelos RBD da Figura 4.15 e CTMCs das Figuras 4.13 e 4.10. A Tabela 6.18 apresenta os parâmetros inseridos no modelo RBD da Figura 4.15.

Tabela 6.18: Parâmetros de entrada do RBD da nuvem com Nós redundantes

Componente	MTTF (h)	MTTR (h)
Frontend	180,72	0,0143
Sub-sistema MBaaS	144,4211	0,0406

Os resultados apresentados na Tabela 6.19 denota uma disponibilidade de 0,997560, com *downtime* que equivale a menos de um dia, dessa forma essa arquitetura demonstra uma melhora considerável na disponibilidade de ambientes MBaaS.

Tabela 6.19: Resultados do modelo RBD redundância Nós e *Frontend*

Métricas	Valores
Disponibilidade	0,999639
Disponibilidade (9's)	3,4435
Downtime (h)	3,1623

A Figura 6.4 apresenta um comparativo entre as infraestruturas baseadas em nuvem. Podemos observar que o estudo de caso referente à arquitetura de nuvem sem redundância apresentou a menor disponibilidade dentre as arquiteturas em nuvem, pois a mesma não implementa nenhum mecanismo de replicação. Ainda analisando o gráfico, observamos que as arquiteturas que implementam mecanismo de redundância obtêm maiores disponibilidades do serviço. Aplicando o mecanismo de redundância nos dois componentes de maior criticidade em uma nuvem, obtivemos os melhores resultados, pois além de alcançar uma disponibilidade de 0,9996, alcançou-se um *downtime* de apenas 3,1623 h.

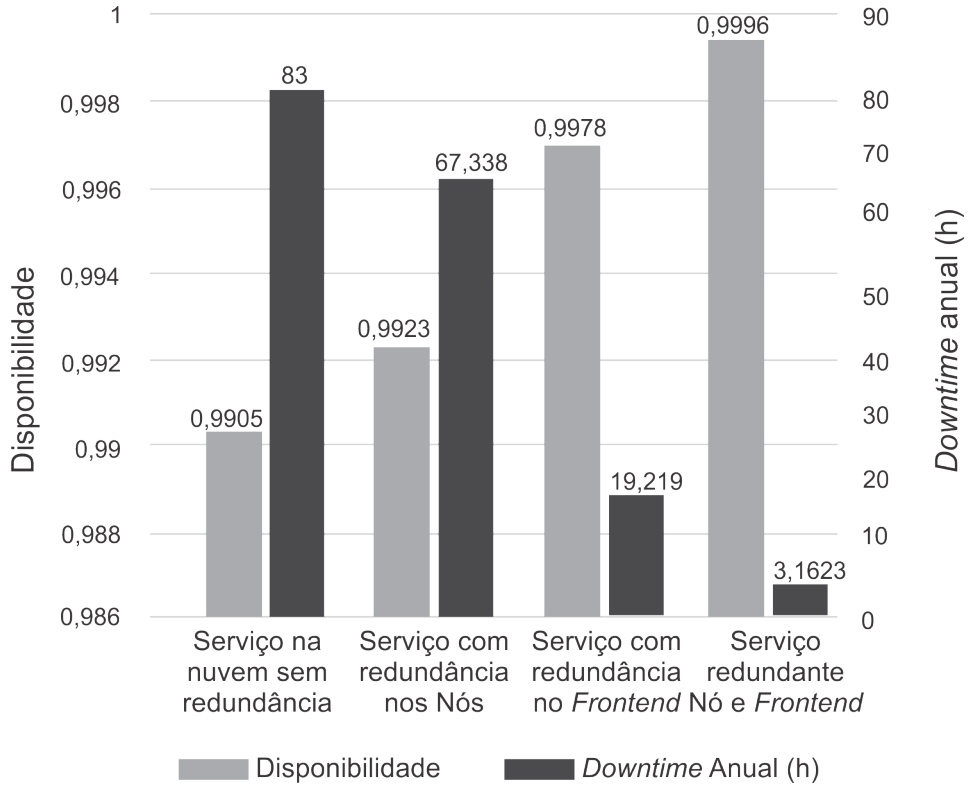


Figura 6.4: Comparativo das arquiteturas de nuvem para MBaaS

6.7 Considerações Finais

Os estudos de caso abordados neste capítulo apresentaram aspectos importantes relacionados à disponibilidade em ambientes MBaaS. Os resultados obtidos oferecem um aporte para que administradores de ambientes MBaaS possam analisar as melhores infraestruturas a serem adotadas do ponto de vista da disponibilidade.

Inicialmente observamos que adotar um processo de recuperação automática no componente de *software* de mais baixo nível da plataforma, que neste caso é a JVM, é uma abordagem que reduz em 10% o *downtime*. Observamos também que na arquitetura básica o componente que causa maior impacto na disponibilidade do serviço é a própria plataforma de MBaaS, pois através de uma análise de sensibilidade esse componente foi identificado como o de maior criticidade.

Com o modelo de serviço consistente e validado, foi possível modelar a plataforma em um ambiente de nuvem privada, onde o estudo de caso que trata o ambiente de nuvem sem redundância demonstrou uma melhora significativa quando aplicado um processo de recuperação automática na plataforma de MBaaS. Porém, dentre os quatros cenários propostos, destacou-se o cenário com redundância no *Frontend* e nos Nós, valendo salientar que o mecanismo de redundância baseado para a geração dos modelos foi o mecanismo *warm standby*, e que a redundância aplicada nos Nós utilizou uma política de manutenção que prioriza o reparo do serviço antes de qualquer outra possibilidade.

7

Conclusão

O uso crescente da computação móvel está se tornando cada vez mais evidente, e de acordo com (STERLING, 2014) 30% do tráfico de internet atual é oriundo de dispositivos móveis, como *tablets* e *smartphones*. Nos últimos anos, os aplicativos voltados para dispositivos móveis tornaram-se abundantes, com aplicações em diversas categorias, como entretenimento, saúde, jogos, negócios, redes sociais, viagens e notícias (FERNANDO; LOKE; RAHAYU, 2013). Esse contexto define um cenário promissor para que empresas possam migrar seus processos e serviço para o ambiente mobile, no entanto, diversos são os desafios nessa área, a exemplo das limitações dos dispositivos móveis, como armazenamento e processamento. A fim de minimizar essas restrições, é possível integrar o ambiente de nuvem com os dispositivos, permitindo, por exemplo, que tanto o armazenamento quanto o processamento aconteçam no lado da nuvem. É nessa conjuntura que atua o ambiente *Mobile Backend as a Service*, uma vez que tornar essa infraestrutura altamente disponível pode determinar a confiança dos usuários em aplicações que utilizem esse ambiente.

Neste âmbito, foi realizado um estudo de avaliação da disponibilidade de ambientes MBaaS, com a utilização da plataforma de MBaaS *open source* OpenMobster. Investigou-se cinco diferentes cenários, a fim de verificar a viabilidade de cada um em relação à disponibilidade. O primeiro cenário analisado foi definido como arquitetura básica, que teve como principal objetivo modelar e validar o serviço da plataforma, como também verificar o comportamento da mesma com a utilização de um processo de recuperação automática no componente JVM. Para isso, uma abordagem utilizando modelagem hierárquica com CTMC e RBD foi concebida, a fim de representar o comportamento do sistema com e sem a utilização desse processo, além disso, fez-se uma análise da disponibilidade de acordo com a variação da probabilidade de sucesso do processo de recuperação. Os resultados apresentaram uma disponibilidade de 89,69% na arquitetura básica sem o processo de recuperação automática e com um *downtime* de 902.76 h. Com o processo de recuperação automática com probabilidade de sucesso de 99% sendo aplicado nesta arquitetura, obteve-se uma disponibilidade de 90,70% e um *downtime* de 814,56 h, e este resultado representa uma redução de 10% no *downtime* em relação à arquitetura básica. Pode-se então inferir que a utilização do processo de recuperação no componente JVM, tem

sim um efeito considerável sobre a disponibilidade do sistema. É importante ressaltar que os modelos da arquitetura básica foram validados através de um experimento de injeção de falhas. Por coerência, os mesmos valores de tempo de falha e reparo que foram reduzidos durante o experimento na injeção de falhas, serviram como insumo para os parâmetros dos modelos avaliados neste estudo de caso.

Com base neste cenário, fez-se uma análise de sensibilidade paramétrica, onde foi possível verificar os gargalos de disponibilidade e propor variações da arquitetura básica com o objetivo de melhorar a disponibilidade do serviço. O resultado da aplicação da análise de sensibilidade nesse ambiente identificou que o OpenMobster é o componente de maior impacto sobre a disponibilidade da infraestrutura, isso sugere que a implementação de redundância nesse componente pode melhorar significativamente a disponibilidade. Assim, sugerimos a modelagem do serviço em uma estrutura de nuvem privada.

O segundo cenário abordado foi caracterizado como arquitetura básica de nuvem privada sem redundância, que foi baseado na plataforma Eucalyptus. Nele propomos a implementação de um processo de recuperação automática na plataforma de MBaaS. Os resultados deste estudo de caso apresentam uma disponibilidade 99,04% e um *downtime* de 83,27 horas. A fim de verificar o impacto do processo de recuperação automática na disponibilidade, refizemos a análise sem o processo e dessa forma obtivemos uma disponibilidade de 97,45% e um *downtime* de 222 horas. Este resultado representa um aumento no *downtime* de 138,73 horas em relação à arquitetura básica. Uma análise de sensibilidade aplicada neste cenário guiou a construção das arquiteturas subsequentes.

O terceiro cenário foi a elaboração de arquitetura de nuvem com redundância *warm standby* no *Frontend*. Para isso, foi utilizado o modelo proposto por (DANTAS et al., 2012). Os resultados apresentam uma disponibilidade de 99,78% e um *downtime* de 19,21 h, observando-se assim uma redução de 77 % em relação ao *downtime* da arquitetura básica de nuvem.

O quarto cenário foi construído com uma arquitetura de nuvem com redundância *warm standby* nos Nós e nessa arquitetura foi proposta uma política de manutenção de priorização do serviço, ou seja, nas situações em que o serviço tiver condições de ser reparada, essa será a prioridade. Os resultados apresentaram uma disponibilidade de 99,23% e um *downtime* de 67,33 h, com uma redução de 19% no *downtime* em relação à arquitetura básica. Por fim, elaboramos uma arquitetura de nuvem com redundância *warm standby* nos Nós e Frontend e foi possível notar uma disponibilidade de 99,96% e *downtime* de 3,16 horas, reduzindo o *downtime* em relação a arquitetura básica em 96%. Esses resultados indicam que a arquitetura com redundância nos Nós e Frontend se mostrou a mais atrativa, pois a mesma obteve os melhores resultados de disponibilidade e o menor *downtime* anual das quatro arquiteturas.

Os resultados obtidos nesse trabalho podem nortear os administradores de ambientes MBaaS na utilização de uma infraestrutura de nuvem semelhante a aqui proposta. Este documento também fornece uma visão geral dessa infraestrutura, bem como seus respectivos modelos analíticos, auxiliando assim na tomada de decisões sobre implementação e políticas de

manutenção.

Desta maneira, estes resultados apresentaram algumas contribuições que podem ser detalhadas da seguinte maneira.

A princípio destacamos uma metodologia de avaliação para ambientes MBaaS. Esta metodologia fez uso de modelagem hierárquica com uso de CTMC e RBD e experimentos de injeção de falhas, com o objetivo de avaliar a efetividade de soluções arquiteturais na melhoria de disponibilidade destes ambientes. Desta forma, esta metodologia pode ser replicada para avaliar a disponibilidade em ambientes com tais características.

Também foi proposto o modelo da arquitetura básica para a plataforma de MBaaS OpenMobster, no qual modelamos a arquitetura básica enfatizando todos os componentes do lado do servidor. Os resultados podem ser utilizados em ambientes mais complexos, tais como ambientes de alta disponibilidade, uma vez que o modelo analítico de disponibilidade aqui produzido foi validado e representa o serviço oferecido pela plataforma de MBaaS.

Desenvolvemos uma análise de sensibilidade na arquitetura de nuvem privada sem redundância no ambiente com uma plataforma de MBaaS implantada, a partir dessa análise é possível verificar os gargalos de disponibilidade nesse ambiente, desta forma os resultados do *ranking* de sensibilidade pode ser utilizado para proposição de arquiteturas baseadas nesse ambiente.

Em decorrência da proposição dos modelos de disponibilidade, também produzimos um conjunto de fórmulas fechadas que podemo ser úteis para verificar a disponibilidade em ambientes MBaaS.

Por fim, este trabalho apresenta uma análise detalhada da disponibilidade de diversas arquiteturas baseadas em nuvem privada, com implantação do serviço de MBaaS. Desta forma, estas arquiteturas, bem como os modelos que as representam podem ser utilizadas em pesquisas futuras.

7.1 Trabalhos Futuros

Após a conclusão dessa pesquisa verificou-se que a mesma pode ser ampliada através de propostas futuras, como por exemplo, realizar a análise de outros atributos de dependabilidade em ambientes de MBaaS, além da disponibilidade estacionária e *downtime* anual. Como uma avaliação de desempenho da plataforma de MBaaS em um ambiente de nuvem privada, a fim de verificar o comportamento de VMs sobre as métricas de memória, CPU e disco, para níveis distintos dos parâmetros como *workload* de requisições de usuários e utilização do disco. Tais métricas são críticas para o provimento de um serviço MBaaS. Os provedores de serviços não só são obrigados a fornecer serviços corretos, mas, também, a satisfazer as expectativas no contexto de desempenho. Desta forma, uma avaliação deste tipo pode auxiliar administradores destes ambientes a prover um serviço com melhor qualidade.

Além da proposta citada, pode-se destacar que ampliar o ambiente de investigação

focando também no lado cliente, e extrair deste, métricas como disponibilidade e desempenho, pode ser um viés interessante para pesquisas futuras envolvendo esta área, pois como citado anteriormente o desempenho é uma importante métrica no provimento de serviços, além disso, neste contexto, é possível analisar o comportamento da aplicação no lado cliente, como por exemplo, verificar a autonomia da bateria com a utilização de uma aplicação baseada em uma plataforma de MBaaS, bem como analisar a disponibilidade com diferentes meios de comunicação. Como também construir um *testbed* da arquitetura com os Nós e *Frontends* redundantes e validar o modelo da mesma, acrescido do lado cliente.

Como o trabalho ora abordado apenas considerou o mecanismo de redundância *warm standby*, analisar outros mecanismos de replicação e extrair desses ambientes métricas de dependabilidade como disponibilidade e desempenho se torna uma abordagem importante, com o objetivo de verificar a eficiência destes mecanismos aplicados no tipo de infraestruturas adotadas nesta pesquisa.

Referências

AERONAUTICS, A. I. of; STAF, A. AIAA Guide for the Verification and Validation of Computational Fluid Dynamics Simulations. , [S.l.], 1998.

APPARAO, P. et al. Characterization & Analysis of a Server Consolidation Benchmark. In: FOURTH ACM SIGPLAN/SIGOPS INTERNATIONAL CONFERENCE ON VIRTUAL EXECUTION ENVIRONMENTS, New York, NY, USA. **Proceedings...** ACM, 2008. p.21–30. (VEE '08).

ARAÚJO, C. J. M. **Avaliação e modelagem de desempenho para planejamento de capacidade do sistema de transferência eletrônica de fundos utilizando tráfego em rajada**. 2009. Tese (Doutorado em Ciência da Computação) — Dissertação, Universidade Federal de Pernambuco, Centro de Informática.

ARAUJO, J. et al. Dependability evaluation of a mhealth system using a mobile cloud infrastructure. In: SYSTEMS, MAN AND CYBERNETICS (SMC), 2014 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014. p.1348–1353.

ARCHER, J. et al. **Security guidance for critical areas of mobile computing**. 2012.

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

ARMBRUST, M. et al. A View of Cloud Computing. **Commun. ACM**, New York, NY, USA, v.53, n.4, p.50–58, Apr. 2010.

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

AVIZIENIS, A.; LAPRIE, J. claude; RANDELL, B. **Fundamental Concepts of Dependability**. [S.l.: s.n.], 2001.

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

BAL, S. N. Mobile web–enterprise application advantages. **International Journal of Computer Science and Mobile Computing**, [S.l.], v.2, n.2, p.36–40, 2013.

BAUER E., A. R.; EUSTACE, D. **Beyond Redundancy**: how geographic redundancy can improve service availability and reliability of computer-based systems. Hoboken, NJ, USA: John Wiley l& Sons, Inc., 2011.

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

BOLCH, G. et al. **Queueing Networks and Markov Chains** : modeling and performance evaluation with computer science applications. [S.l.]: Wiley & Sons, New York, NY, USA, 1998.

BOLCH, G. et al. **Queuing Networks and Markov Chains**: modeling and performance evaluation with computer science applications. 2.ed. [S.l.]: John Wiley and Sons, 2001.

BOYLE, C. **Smartphone users worldwide will total 1.75 billion in 2014**. 2014.

BRILHANTE, J. et al. Eucabomber 2.0: a tool for dependability tests in eucalyptus cloud infrastructures considering vm life-cycle. In: SYSTEMS, MAN AND CYBERNETICS (SMC), 2014 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014. p.2669–2674.

CAMPOS, E. et al. Performance Evaluation of Virtual Machines Instantiation in a Private Cloud. , [S.l.], 2015.

CAROLAN, J.; GAEDE, S. Introduction to cloud computing architecture. , [S.l.], 2009.

ČEPIN, M. **Assessment of Power System Reliability**: methods and applications. [S.l.]: Springer Science & Business Media, 2011.

COOPER, B. F. et al. Benchmarking Cloud Serving Systems with YCSB. In: ACM SYMPOSIUM ON CLOUD COMPUTING, 1., New York, NY, USA. **Proceedings...** ACM, 2010. p.143–154. (SoCC '10).

DANTAS, J. Modelos para Análise de Dependabilidade de Arquiteturas de Computação em Nuvem. , [S.l.], 2013.

DANTAS, J. et al. An Availability Model for Eucalyptus Platform: an analysis of warm-standy replication mechanism. , [S.l.], v.3, n.3, p.1664–1669, October 2012.

DILLON, T.; WU, C.; CHANG, E. Cloud Computing: issues and challenges. In: IEEE INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION NETWORKING AND APPLICATIONS, 2010. **Anais...** IEEE, 2010.

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

EJLALI, A. R. et al. A Hybrid Fault Injection Approach Based on Simulation and Emulation Co-operation. , [S.l.], p.479–488, 2003.

EUCALYPTUS. **Eucalyptus Open-Source Cloud Computing Infrastructure - An Overview**. [S.l.]: Eucalyptus Systems, Inc., Goleta, CA, 2009.

EUCALYPTUS. **Eucalyptus - the open source cloud platform**. Disponível em `http://open.eucalyptus.com/`, acessado em Dez 2014.

FERNANDO, N.; LOKE, S. W.; RAHAYU, W. Mobile cloud computing: a survey. **Future Generation Computer Systems**, [S.l.], v.29, n.1, p.84 – 106, 2013. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

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

FURTH, B.; ESCALANTE, A. Handbook of cloud computing. , [S.l.], 2010.

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

HA PROJECT linux. **Heartbeat**. Disponível em `http://www.linuxha.org`, acessado em Dez 2014.

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

HELLER, M. **MBaaS shoot-out**: 5 clouds for building mobile apps. Disponível em `http://goo.gl/l3mtru`, acessado em Nov 2014.

HOANG, D. T. et al. A survey of mobile cloud computing: architecture, applications, and approaches. **Wireless Communications and Mobile Computing**, [S.l.], v.13, n.18, p.1587–1611, 2013.

HSQLDB. **2.3.2 released!** Disponível em `http://hsqldb.org/`, acessado em Dez 2014.

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

JBOSS. **Jboss**. Disponível em `http://www.jboss.org/`, acessado em Dez 2014.

KEESEE, W. R. A Method of Determining a Confidence Interval for Availability. , Point Mugu, California, 1965.

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

KIM, D. S.; MACHIDA, F.; TRIVEDI, K. S. Availability modeling and analysis of a virtualized system. **Dependable Computing, 2009. PRDC '09. 15th IEEE Pacific Rim International Symposium on**, [S.l.], p.365–371, 2009.

KUMAR, K.; LU, Y.-H. Cloud Computing for Mobile Users: can offloading computation save energy? **Computer**, [S.l.], v.43, n.4, p.51–56, April 2010.

KUO, W.; MING, Z. **Optimal Reliability Modeling: principles and applications**. [S.l.]: Wiley, 2002.

LANE, K. **Rise of Mobile Backend as a Service (MBaaS) API Stacks**. 2012.

LOGOTHETIS, D.; TRIVEDI, K. Time-Dependent Behavior of Redundant Systems with Deterministic Repair. In: STEWART, W. (Ed.). **Computations with Markov Chains**. [S.l.]: Springer US, 1995. p.135–150.

LONGO, F. et al. A scalable availability model for Infrastructure-as-a-Service cloud. In: DEPENDABLE SYSTEMS NETWORKS (DSN), 2011 IEEE/IFIP 41ST INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.335–346.

LYON; FYODOR, G. **Nmap Network Scanning**: the official nmap project guide to network discovery and security scanning. USA: Insecure, 2009.

MACIEL, P. et al. Performance and Dependability in Service Computing: concepts, techniques and research directions. , [S.l.], v.1, n.1, p.53–97, 2011.

MACIEL, P. R. M. et al. Dependability Modeling. , [S.l.], p.53 –97, 2012.

MALHOTRA, M.; TRIVEDI, K. A methodology for formal expression of hierarchy in model solution. In: PETRI NETS AND PERFORMANCE MODELS, 1993. PROCEEDINGS., 5TH INTERNATIONAL WORKSHOP ON. **Anais...** [S.l.: s.n.], 1993. p.258–267.

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.

MARKETSANDMARKETS. World Mobile Applications Market - Advanced Technologies, Global Forecast (2010 - 2015). , [S.l.], 2010. Disponível em `http://migre.me/qRuG7`, acessado em Set 2014.

MARKETSANDMARKETS.COM. Cloud Backend-as-a-service (BaaS)/ Mobile BaaS (MBaaS) Market - Global Advancements, Business Models, Technology Roadmap, Forecasts Analysis (2012-2017). , [S.l.], 2012. Disponível em `http://migre.me/qRtZw`, acessado em Dez 2014.

MARWAH, M. et al. Quantifying the Sustainability Impact of Data Center Availability. **SIGMETRICS Perform. Eval. Rev.**, New York, NY, USA, v.37, n.4, p.64–68, Mar. 2010.

Matos Junior, R. **An automated approach for systems performance and dependability improvement through sensitivity analysis of Markov chains**. 2011.

MATOS, R. et al. Sensitivity analysis of server virtualized system availability. **Reliability, IEEE Transactions on**, [S.l.], v.61, n.4, p.994–1006, 2012.

MATOS, R. et al. Sensitivity analysis of a hierarchical model of mobile cloud computing. **Simulation Modelling Practice and Theory**, [S.l.], n.0, p.–, 2014.

MATOS, R.; MACIEL, P. An automated approach for systems performance and dependability improvement through sensitivity analysis of markov chains. **Environmental Monitoring and Assessment**, [S.l.], 2011.

MELL, P.; GRANCE, T. The NIST definition of cloud computing (draft). In: NIST SPECIAL PUBLICATION. **Anais...** [S.l.: s.n.], 2011. v.800, n.145, p.7.

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

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

MENASCE, D. A.; DOWDY, L. W.; ALMEIDA, V. A. F. **Performance by Design**: computer capacity planning by example. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

MOLLOY, M. K. Performance Analysis Using Stochastic Petri Nets. **IEEE Trans. Comput.**, Washington, DC, USA, v.31, n.9, p.913–917, Sept. 1982.

NICOL, D.; SANDERS, W.; TRIVEDI, K. Model-based evaluation: from dependability to security. **Dependable and Secure Computing, IEEE Transactions on**, [S.l.], v.1, n.1, p.48–65, Jan 2004.

O'CONNOR, P.; O'CONNOR, P.; KLEYNER, A. **Practical Reliability Engineering**. [S.l.]: Wiley, 2012. (Quality and Reliability Engineering Series).

Oliveira, D. **Análise de Disponibilidade e Consumo Energético em Ambientes de Mobile Cloud Computing**. 2011.

OLIVEIRA, D. et al. Availability and Energy Consumption Analysis of Mobile Cloud Environments. , [S.l.], p.4086–4091, 2013.

OPENMOBSTER. **OpenMobster - Open Source Mobile Enterprise Backend**. Disponível em `http://code.google.com/p/openmobster`, acessado em Set 2014.

OPENMOBSTER. **OpenMobster - Developer guide**. Disponível em `http://goo.gl/ILTvYb`, acessado em Nov 2014.

PANDEY, S.; NEPAL, S. Modeling Availability in Clouds for Mobile Computing. In: MOBILE SERVICES (MS), 2012 IEEE FIRST INTERNATIONAL CONFERENCE ON. **Anais...** IEEE, 2012. p.80–87.

PARHAMI, B. From Defects to Failures: a view of dependable computing. **SIGARCH Comput. Archit. News**, New York, NY, USA, v.16, n.4, p.157–168, Sept. 1988.

PROJECT, R. **The R Project for Statistical Computing**. Disponível em `https://www.r-project.org/`, acessado em Dez 2014.

QI, H.; GANI, A. Research on mobile cloud computing: review, trend and perspectives. In: SECOND INTERNATIONAL CONFERENCE ON DIGITAL INFORMATION AND COMMUNICATION TECHNOLOGY AND IT'S APPLICATIONS (DICTAP), 2012. **Anais...** IEEE, 2012.

QURESHI, S. S. et al. Mobile cloud computing as future for mobile applications - Implementation methods and challenging issues. , [S.l.], p.467–471, 2011.

ROUSE, M. **Mobile Backend as a Services**. 2015.

ROWINSKI, D. **The Rise of Mobile Cloud Services**: baas startups grow up. 2012.

SANAEI, Z. et al. SAMI: service-based arbitrated multi-tier infrastructure for mobile cloud computing. , [S.l.], 2012.

SAREEN, P. Cloud Computing: types, architecture, applications, concerns, virtualization and role of it governance in cloud. **International Journal of Advanced Research in Computer Science and Software Engineering**, Washington, DC, USA, v.3, n.3, p.533–538, March 2013.

SARGENT, R. G. Verification And Validation Of Simulation Models. , [S.l.], p.55–64, 1998.

SATYANARAYANAN, M. et al. The Case for VM-Based Cloudlets in Mobile Computing. **IEEE Pervasive Comput.**, [S.l.], v.8, n.4, p.14–23, oct 2009.

SIEGERT, S.; FRIEDRICH, R.; PEINKE, J. Analysis of data sets of stochastic systems. **arXiv preprint cond-mat/9803250**, [S.l.], 1998.

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

SILVA, B. et al. Mercury: an integrated for performance and dependability evaluation of general system. In: INDUSTRIAL TRACK AT 45TH DEPENDABLE SYSTEMS AND NETWORK CONFERENCE (DSN 2015). **Proceedings...** [S.l.: s.n.], 2015.

SOUSA, E. Teixeira Gomes de; ROMERO MARTINS MACIEL, P. O. Avaliação do impacto de uma política de manutenção na performabilidade de sistemas de transferência eletrônica de fundos. , [S.l.], 2009.

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

SOUZA, D. et al. A Tool for Automatic Dependability Test in Eucalyptus Cloud Computing Infrastructures. , Point Mugu, California, v.6, n.3, 2013.

STERLING, G. **Mobile Devices**: 30 percent of traffic, 15 percent of sales. Disponível em `http://goo.gl/mPjzbH`, acessado em Set 2014.

SUNMICROSYSTEMS. **Introduction to Cloud Computing Architecture**. [S.l.]: SunMicrosystems, Inc., 1 edition, 2009.

TEOREY, T. J.; NG, W. T. Dependability and Performance Measures for the Database Practitioner. **IEEE Trans. Knowl. Data Eng.**, [S.l.], v.10, n.3, p.499–503, 1998.

TRIVEDI, K.; PULIAFITO, A. Performance And Reliability Analysis Of Computer Systems (an Example-based Approach Using The Sharpe Software. **Reliability, IEEE Transactions on**, [S.l.], v.46, n.3, p.441–441, Sep 1997.

TRIVEDI, K. S. **Probability and Statistics with Reliability, Queuing and Computer Science Applications**. 2nd edition.ed. Chichester, UK: John Wiley and Sons Ltd., 2002.

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

VACARO, J. C.; WEBER, T. S. Injeção de Falhas na Fase de Teste de Aplicações Distribuídas. , [S.l.], 2006.

VAQUERO, L. M. et al. A Break in the Clouds: towards a cloud definition. **SIGCOMM Comput. Commun. Rev.**, New York, NY, USA, v.39, n.1, p.50–55, Dec. 2008.

WEYNS, K.; HÖST, M. Case Study on Risk Analysis for Critical Systems with Reliability Block Diagrams. In: INTERNATIONAL IT SYSTEMS FOR CRISIS RESPONSE AND MANAGEMENT (ISCRAM) CONFERENCE, 10. **Anais...** [S.l.: s.n.], 2013.

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

ZHANG, Y.; LIU, B.; ZHOU, Q. A dynamic software binary fault injection system for real-time embedded software. In: RELIABILITY, MAINTAINABILITY AND SAFETY (ICRMS), 2011 9TH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.676–680.

ZIADE, H. et al. A survey on fault injection techniques. **Int. Arab J. Inf. Technol.**, [S.l.], v.1, n.2, p.171–186, 2004.

Apêndice

A

Scripts de Injeção de falha do Jboss

```
#!/bin/bash


echo "Fault injection Jboss...."
echo "# Event GT Date" >> JbossFault.log

C1=0

while [ True ]
do

        C1=$(($C1+1))
        STATUS=$(nmap -p 80 192.168.0.181 | grep 80/tcp | awk '{print
            $2}')
        VERIFICA="0"
        #Verifica se o Jboss est  ativo
        if [ "$STATUS" == "open" ]
        then

                TIME=$(Rscript expFailJboss.r | awk '{print $2}')
                echo "Wait time to failure = " $TIME
                sleep $TIME

                echo "Fault injected"
                echo $C1 "Fault" $TIME $(date | awk '{ print $2, $3,
                    $4 }') >> JbossFault.log
                VERIFICA="1"

        fi
        #Se VERIFICA for igual, a falha foi injetada corretamente
        if [ "$VERIFICA" == "1" ]
        then

                TIMER=$(Rscript expRepairJboss.r | awk '{print $2}' |
```

```
                awk -F"." '{print $1}')
33              while [ $TIMER -lt 10 ] ;
34              do
35                      echo "time less than 10..."
36                      TIMER=$(Rscript expRepairJboss.r | awk '{
                            print $2}' | awk -F"." '{print $1}')
37
38              done
39
40              echo "time to repair: " $TIMER
41              sleep $TIMER
42              STATUS=$(nmap -p 80 192.168.0.181 | grep 80/tcp | awk
                    '{print $2}')
43              C2=0
44
45      while [ "$STATUS" == "closed" ]
46      do
47
48              nohup sshpass -p "modcs" ssh root/192.168.0.181 "/etc
                    /init.d/jboss5 start" & 2>1& /dev/null
49              nohup sshpass -p "modcs" ssh root/192.168.0.181 "
                    iptables -I INPUT -p tcp --dport 80 -j REJECT"
50              sleep 10
51              C2=$(($C2+10))
52              STATUS=$(nmap -p 80 192.168.0.181 | grep 80/tcp | awk
                    '{print $2}')
53
54              echo "Trying start up Jboss...."
55
56      done
57
58              echo $C1 "Subtracting time" $C2 "s" >> JbossFault.log
59              echo $C1 "RepairJboss" $TIMER $(date | awk '{ print
                    $2, $3, $4 }') >> JbossFault.log
60
61              echo "Repair Injected"
62
63              # =============== Tempo de Reparo para o OpenMobster
                    ========================
64
65              TIMER=$(Rscript expRepairJboss.r | awk '{print $2}')
66              while [ $TIMER -lt 10 ] ;
67              do
68
69                      echo "time less than 10..."
70                      TIMER=$(Rscript expRepairJboss.r | awk '{
                            print $2}' | awk -F"." '{print $1}')
```

```
71
72                  done
73
74                  echo "time to repair: " $TIMER
75                  sleep $TIMER
76                  nohup sshpass -p "modcs" ssh root/192.168.0.181 "
                        iptables -F"
77                  echo $C1 "RepairOpen" $TIMER $(date | awk '{ print $2
                        , $3, $4 }') >> OpenMobsterFault.log
78                  echo "Repair Open Injected"
79
80          fi
81
82  done
83
84  echo "Terminou Node"
```

B

Scripts de Injeção de falha da JVM

```
1  #!/bin/bash
2
3
4  echo "Fault injection Java...."
5  echo "# Event GT Date" >> JbossFault.log
6
7  C1=0
8
9  while [ True ]
10 do
11         C1=$(($C1+1))
12         TIME=$(Rscript expFailJboss.r | awk '{print $2}')
13         echo "Wait time to failure = " $TIME
14         sleep $TIME
15
16         # ============== Exclus o M tua ========================
17
18         FJBOSS=$(ps -A |grep Fail_Jboss |awk '{print $1}')
19         FOPEN=$(ps -A |grep OpenMobsterFault |awk '{print $1}')
20         kill -STOP $FJBOSS
21         kill -STOP $FOPEN
22
23         # ============== Fim Exclus o M tua
               ========================
24
25         nohup sshpass -p "modcs" ssh root/192.168.0.181 "sh /opt/
               jboss-5.1.0.GA/bin/shutdown.sh -S" & 2>1& /dev/null
26         sshpass -p "modcs" ssh root/192.168.0.181 "sh /usr/script/
               failjboss.sh"
27
28         echo "Fault injected"
29         echo $C1 "Fault injected Java" $TIMER $(date | awk '{ print
               $2, $3, $4 }') >> JavaFault.log
30         TIMER=$(Rscript expRepairJboss.r | awk '{print $2}' | awk -F"
```

```
            ." '{print $1}')
31
32          while [ $TIMER -lt 10 ] ;
33          do
34
35          echo "time less than 20..."
36          TIMER=$(Rscript expRepairJboss.r | awk '{print $2}' | awk -F"
                ." '{print $1}')
37
38          done
39
40          echo "time to repair: " $TIMER
41          sleep $TIMER
42          nohup sshpass -p "modcs" ssh root/192.168.0.181 "sh /opt/
                jboss-5.1.0.GA/bin/run.sh" & 2>1& /dev/null
43          echo $C1 "RepairJava" $TIMER $(date | awk '{ print $2, $3, $4
                 }') >> JavaFault.log
44
45          # ============== Libera  o das Threads
                ========================
46
47          kill -CONT $FJBOSS
48          kill -CONT $FOPEN
49
50          # ============== Fim Libera  o das Threads
                ====================
51  done
52  echo "Terminou Node"
```

C

Scripts de Monitoramento da disponibilidade da Plataforma de MBaaS

```
1  #!/bin/bash
2
3  C=0
4
5  echo "Start" $(date | awk '{ print $2, $3, $4 }') >> "TesteAvailOpen"
       .txt
6  echo "# Port StatusPC StatusOpenMobster Status "  >> "TesteAvailOpen"
       .txt
7
8  while [ True ]
9  do
10         C=$(($C+1))
11         STATUS=$(nmap -p 80 192.168.0.181 | grep 80/tcp | awk '{print
               $2}')
12         STATUSS=$(nmap -p 80 192.168.0.181 | grep "Nmap done" | awk
              '{print $6}')
13
14         if [ "$STATUSS" == "(1" ] ; then
15                 if [ "$STATUS" == "open" ] ; then
16                         echo $C $STATUS "up" "J/O up" $(date | awk '{
                               print $2, $3, $4 }') >> "TesteAvailOpen".
                              txt
17                 else
18                         echo $C $STATUS "up" "J/O down" $(date | awk
                              '{ print $2, $3, $4 }') >> "TesteAvailOpen
                              ".txt
19                 fi
20         else
21                 echo $C "Offline" "down" "J/O down" $(date | awk '{
                       print $2, $3, $4 }') >> "TesteAvailOpen".txt
22         fi
```

```
23          sleep 5
24  done
25  echo "End" $(date | awk '{ print $2, $3, $4 }') > "TesteAvailOpen".
        txt
```

D

Scripts de Monitoramento de inicialização do Jboss

```
#!/bin/bash

echo "====Initializacion openmobster===="
echo "# Event GT Date" >> timeinitializacion.log
C1=0
while [ True ]
do
        C1=$(($C1+1))

        STATUS=$(nmap -p 80 192.168.0.184 | grep 80/tcp | awk '{print
            $2}')

        if [ "$STATUS" == "closed" ]
        then
                sleep 20
                TIMER=$(./run.sh -c openmobster -b 192.168.0.184 |
                    grep Started)
                sleep 40

                echo $C1 "TIME" $TIMER >> timeinitializacion.log
        fi

        if [ "$STATUS" == "open" ]
        then
                        exit 0
        fi

done
echo "Terminou Node"
```