Pós-Graduação em Ciência da Computação

Jean Carlos Teixeira de Araujo

PLANEJAMENTO DE INFRAESTRUTURAS DE *MOBILE CLOUD COMPUTING* BASEADO EM MODELOS ESTOCÁSTICOS

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

RECIFE-PE

2017

Jean Carlos Teixeira de Araujo

PLANEJAMENTO DE INFRAESTRUTURAS DE *MOBILE CLOUD COMPUTING* BASEADO EM MODELOS ESTOCÁSTICOS

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 Doutor em Ciência da Computação.

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

RECIFE-PE
2017

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

A663p Araujo, Jean Carlos Teixeira de
Planejamento de infraestruturas de *mobile cloud computing* baseado em modelos estocásticos / Jean Carlos Teixeira de Araujo. – 2017.
149 f.: il., fig., tab.

Orientador: Paulo Romero Martins Maciel.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2017.
Inclui referências e apêndices.

1. Ciência da computação. 2. Computação em nuvem. 3. Modelos estocásticos. I. Maciel, Paulo Romero Martins (orientador). II. Título.

004 CDD (23. ed.) UFPE- MEI 2017-120

Jean Carlos Teixeira de Araujo

Planejamento de Infraestruturas de *Mobile Cloud Computing* Baseado em Modelos Estocásticos

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutora em Ciência da Computação.

Aprovado em: 09/03/2017.

_____

Orientador: Prof. Dr. Paulo Romero Martins Maciel

BANCA EXAMINADORA

_____

Prof. Dr. Djamel Fawzi Hadj Sadok
Centro de Informática / UFPE

_____

Prof. Dr. Nelson Souto Rosa
Centro de Informática / UFPE

_____

Prof. Dr. Eduardo Antônio Guimarães Tavares
Centro de Informática / UFPE

_____

Prof. Dr. José Neuman de Souza
Departamento de Computação/UFC

_____

Prof. Dr. Edmundo Roberto Mauro Madeira
Instituto de Computação/UNICAMP

Dedico esta tese a José Domingos (in memoriam), que foi o meu exemplo de pai e avô. Espero ter sido merecedor de todo o seu amor, carinho e dedicação, e honrá-lo com a obtenção do título de doutor.

Agradecimentos

Antes de mais nada, gostaria de agradecer ao bom Deus por ter me dado os intrumentos necessários para chegar até aqui, e por toda a persistência para não desistir diante das dificuldades encontradas no caminho.

Agradecer também ao Prof. Paulo Maciel, que acreditou em meu potencial, me orientou durante toda essa jornada e ajudou em meu crescimento pessoal e profissional, o qual eu tenho uma eterna dívida de gratidão.

Agradecer também à toda a minha família, que sempre estiveram comigo. Em especial à minha mãe Josinete, a qual amo incondicionalmente, presente em todos os momentos da minha vida, por me ensinar a lutar e nunca destir dos meus objetivos.

Muito obrigado a todos os companheiros do grupo de pesquisa MoDCS. Em especial a Rubens, amigo para todos os desafios, pelas valiosas dicas e por estar sempre disposto a ajudar. A Jamilson e Renata, pelo apoio e por não me deixarem dormir na rua por muitas vezes. E a todos os colegas que me ajudaram durante toda essa jornada: Gabriel Alves, Danilo Oliveira, João Ferreira, Verônica Conceição e Ronierison Maciel.

Também sou muito grato a Cristiano e Nira por me receberam sempre de portas abertas em sua casa. Vocês também são a minha família. Meu sincero agradecimento também aos meus ex-orientandos Céfanys e Paulo Roberto que também doaram um pouco do seu tempo.

Não há nada nobre em ser superior ao seu semelhante. A verdadeira nobreza é ser superior ao seu antigo eu.

—ERNEST HEMINGWAY

Resumo

A academia e a indústria têm demonstrado que os recursos limitados dos dispositivos móveis podem ser complementados por recursos virtualizados em uma infraestrutura de computação em nuvem, surgindo assim um novo paradigma chamado *mobile cloud computing* (MCC). Embora esta solução expanda substancialmente a capacidade de tais dispositivos, também impõe uma dependência em tempo integral de conexão de Internet sem fio. Além disso, problemas como o esgotamento da carga da bateria, falhas de dispositivos móveis, instabilidade das redes sem fio, *bugs* de aplicativos e interrupções no serviço da nuvem podem representar obstáculos na expansão da *mobile cloud computing*, uma vez que tais problemas podem resultar no mal fornecimento do serviço da MCC. Modelos estocásticos hierárquicos são adequados para descrever de forma concisa o funcionamento de uma infraestrutura de MCC, lidando com o grande número de componentes que constituem esse tipo de sistema. Sendo um paradigma tão recente, poucos esforços foram feitos para identificar os impactos destes tipos de falhas sobre o funcionamento do sistema. Desta forma, esta tese provê modelos estocásticos para o planejamento de infraestruturas de *mobile cloud computing*. Esta abordagem foca especialmente no impacto de alguns fatores sobre a disponibilidade e confiabilidade do sistema, tais como: arquiteturas distintas de *mobile cloud computing*; características de protocolos de comunicação; componentes críticos dos dispositivos móveis, como a bateria; e o comportamento dos *softwares* envolvidos. A avaliação adotada é baseada em modelos heterogêneos hierárquicos e se concentra em métricas como a disponibilidade em estado estacionário, confiabilidade, *downtime*, probabilidade de conectividade, tempo médio de vida da bateria, e os custos de implantação e provisionamento da infraestrutura de nuvem. Os experimentos realizados fornecem subsídios aos parâmetros de entrada dos modelos aqui propostos. Além disso, ferramentas de *software* foram desenvolvidas para auxiliar na condução de tais experimentos. Os modelos propostos permitem realizar comparações entre diferentes formas de implantação de *mobile cloud computing*, assim como planejar ajustes na infraestrutura de *hardware* e *software* com o intuito de garantir níveis de serviço satisfatórios.

Palavras-chave: *Mobile cloud computing*. Modelos estocásticos. Planejamento de infraestruturas. Tempo de vida da bateria.

Abstract

Academy and industry have been showing that the limited resources of mobile devices might be supplemented by virtualized resources in a cloud infrastructure, emerging a new paradigm called mobile cloud computing (MCC). While this solution substantially expands the abilities of such gadgets, it also enforces a full-time dependency on wireless Internet connection. Furthermore, issues such as battery charge depletion, mobile device faults, wireless network instability, application bugs, and outages in the cloud service may represent obstacles in expansion of the mobile cloud computing, since such issues may result in poor provision of the MCC service. Hierarchical stochastic models are suitable for a concise description of the operation of an MCC infrastructure, dealing with the large number of components that constitute this type of system. Being such a recent paradigm, few efforts were conduced to identify the impact of those types of faults on the system operation. In this way, this thesis provides stochastic models to planning of mobile cloud computing infrastructures. This approach focus specially on the impact of some factors on system availability and reliability, such as: different architectures of mobile cloud computing; characteristics of communication protocols; critical components of mobile devices, such as the battery; and the behavior of the software involved. Our evaluation is based on hierarchical heterogeneous models and focuses on measures such as steady-state availability, reliability, downtime, connectivity probability, average battery lifetime, and costs of implementation and provisioning of the cloud infrastructure. We performed experiments to provide subsidies to the input parameters of the proposed models here. In addition, software tools have been developed to aid in the conduct of such experiments. The proposed models allow comparisons between different forms of mobile cloud computing deployment, as well as planning adjustments to the hardware and software infrastructure to ensure satisfactory service levels.

Keywords: Mobile cloud computing. Stochastic models. Infrastructure planning. Battery lifetime.

Lista de figuras

Lista de tabelas

Lista de Acrônimos

MRM *Markov Reward Models* 72

MTBF *Mean Time Between Failures* 27

MTTA *Mean Time to Activate* 60

MTTARF *Mean Time to Aging-related Failure* 98

MTTF *Mean Time to Failure* 27

MTTR *Mean Time to Repair* 27

MTTRA *Mean Time to Rejuvenation Action* 98

MV Máquina Virtual 59

MVC *Model View Controller* 143

PaaS *Platform as a Service* 24

NAS *Network-Attached Storage* 67

NC Nó Central 136

ND Nó Distribuído 136

RBD *Reliability Block Diagram* 28

RI *Reliability Importance* 80

SaaS *Software as a Service* 24

SGBD Sistema de Gerenciamento de Banco de Dados 67

SO Sistema Operacional 59

SPN *Stochastic Petri Nets* 29

S3 *Simple Storage Service* 128

SC *Storage Controller* 127

TI Tecnologia da Informação 63

TMR *Triple Modular Redundancy* 60

TTARF *Time To Aging-Related Failure* 36

UMC Utilização de Memória Crítica 94

UML *Unified Modeling Language* 144

VM *Virtual Machine* 128

WLAN *Wireless Local Area Network* 55

Sumário

1

Introdução

Quando eu pensei que não podia continuar, eu me forcei a continuar. Meu sucesso é baseado na persistência, não na sorte!

—ESTÉE LAUDER

Os avanços na computação móvel vem mudando o panorama de serviços de Internet. Um número crescente de usuários tem adotado dispositivos móveis como principais pontos de entrada para o mundo *online*. Esta expansão é evidenciada pelo tráfego de dados de *smartphones* e *tablets*, o qual houve um crescimento de cerca de 74% em 2015, e com previsão de aumento de 30,6 exabytes por mês até 2020, um aumento de oito vezes maior em relação a 2015 (CISCO, 2016). Além disso, há estudos que estimam que 43% da população mundial possui algum tipo de *smartphone*, sendo mais comuns em países da Europa (a exemplo de Espanha com 71% e Reino Unido com 68%) e Estados Unidos (72%) do que em países em desenvolvimento (POUSHTER, 2016). Com isso, o desenvolvimento de aplicações para dispositivos móveis são alavancadas pela concorrência entre as grandes empresas do segmento, além da constante necessidade de melhorias na velocidade e cobertura de redes sem fio.

Paralelamente ao crescimento da computação móvel, a computação em nuvem também tem ganhado destaque e popularidade nos últimos anos. A computação em nuvem fornece recursos computacionais, como servidores, redes, armazenamento e aplicações, de forma flexível e ágil. Uma série de recursos estão disponíveis para o usuário sob demanda com pouco esforço de gerenciamento ou interação com um prestador de serviços (MELL; GRANCE, 2011). Em contrapartida, a computação móvel sofre com a escassez de recursos, mesmo em dispositivos mais modernos. Quando comparado com computadores pessoais, os dispositivos móveis têm uma série de restrições, como provisionamento de energia limitado e baixo poder computacional. Além disso, a instabilidade das redes sem fio também podem ser um problema para esses dispositivos.

A grande variedade de recursos de dispositivos móveis ajuda a trazer avanços em diversas áreas, como na medicina, segurança pública, justiça e os mais diversos tipos de empresas

comerciais.

1.1 Motivação e justificativa

A fim de garantir a satisfação dos usuários, provedores de serviços de *mobile cloud computing* devem garantir requisitos de qualidade de serviço, como disponibilidade e confiabilidade. A capacidade da bateria e a estabilidade da rede podem ser preocupações-chave relacionadas à dependabilidade no lado do cliente, enquanto tolerância a falhas de *hardware* e *software* normalmente merecem maior atenção no lado do servidor.

Em ambientes de *mobile cloud computing* e até mesmo no uso doméstico de computadores, o gerenciamento de recursos é importante. No uso doméstico a falta de recursos computacionais pode gerar insatisfação e levar o usuário a se adaptar com a falta do recurso, mas já na *mobile cloud computing*, a falta de recursos computacionais pode gerar um prejuízo enorme para as empresas. Devido à demanda volátil desses ambientes, o gerenciamento é algo imprescindível para o bom funcionamento do serviço (ULLRICH; LäSSIG; GAEDKE, 2016). Usar estratégias de predições, apesar de não serem exatas, podem ajudar o administrador desses ambientes a entender melhor a forma com que a mudança está ocorrendo, e assim tomar decisões de gerência que visam otimizar os serviços prestados. De toda forma, a infraestrutura precisa estar preparada para a demanda de recursos exigida. Caso não esteja, a qualidade do serviço pode ser comprometida, e em casos piores até suspensa temporariamente, o que além de gerar insatisfação, gera prejuízo para o dono do sistema e aqueles que o usam (BASNEY; LIVNY, 2000). Há diversos métodos para fazer predições desses recursos, porém como o monitoramento desses ambientes coletam dados sobre o tempo, isso faz com que técnicas como séries temporais sejam boas candidatas para fazer predições (KAYS; REHTANZ, 2016).

Um dos fatores relacionados ao gerenciamento de recursos que pode influenciar negativamente na qualidade do serviço é o envelhecimento de *software*, pois se trata de um fenômeno que degrada o sistema ao longo de sua vida operacional, afetando principalmente a disponibilidade e o desempenho dos sistemas. A ocorrência do envelhecimento de *software* em sistemas onde há a junção de vários componentes de *software* pode ser catastrófico, especialmente em ambientes de *mobile cloud computing* que permite a existência de sistemas de diversas aplicações. Quanto mais componentes de *software* houver, maior o risco de ocorrência de falhas causadas pelo envelhecimento. No entanto, uma ação proativa pode ser desencadeada para minimizar os efeitos do envelhecimento *software*, conhecida como rejuvenescimento de *software*. Esta ação é comumente acionada por estratégias baseadas no tempo, no limiar, ou com base em predição (ARAUJO et al., 2011).

De maneira geral, a análise de dependabilidade pode ser feita de duas formas: através de medições no sistema real e através do desenvolvimento de modelos estocásticos. A primeira abordagem é mais custosa e menos desejável, uma vez que este tipo de análise é tipicamente feita de maneira a esperar que qualquer problema ocorra, o que pode representar um problema

em aplicações que já estejam em produção[1]. Por outro lado, a análise por modelagem pode ser feita de forma prévia e também é muito menos custosa (OMIDI; MORADI, 2012).

Modelos estocásticos são largamente usados para modelagem em várias áreas. Dentro de TI, esses métodos já foram aplicados na avaliação de dependabilidade em: *web services* (OMIDI; MORADI, 2012), computação em nuvem (camada de IaaS) (SOUSA et al., 2014), *mobile cloud computing* (MATOS et al., 2015), *clusters* virtuais (WEI; LIN; KONG, 2011), *data centers* (CALLOU et al., 2012), redes inteligentes (*smart grid*) (XIANG; TAUCH; LIU, 2014), infraestruturas de redes (GUIMARAES et al., 2011), rejuvenescimento de *software* em sistemas VoIP (KOUTRAS; SALAGARAS; PLATIS, 2009), agendamento da ação de rejuvenescimento *software* em ambientes de computação em nuvem (MELO et al., 2013), entre outros.

O público-alvo desta tese são os provedores de infraestruturas de *mobile cloud computing* e administradores que necessitam idenficar o impacto que diferentes cenários, parâmetros e componentes podem influenciar na disponibilidade e confiabilidade do sistema. A partir desta tese, eles terão insumos para identificar facilmente a partir do deste documento, ou ainda poderão adaptar os modelos para questões específicas da sua infraestrutura.

1.2 Problemática

Em geral, os recursos de *hardware* móveis foram evoluindo rapidamente, mas alguns componentes ainda são um fator limitante para proporcionar uma melhor disponibilidade para os dispositivos móveis. Enquanto os *smartphones*, as *smarthomes* e até mesmo os dispositivos inteligentes (*wearables*) estão crescendo cada vez mais, eles ainda estão limitados pela capacidade da bateria. De fato, as baterias não mantiveram um crescimento no mesmo ritmo que outros componentes de *hardware*. Existem várias tecnologias nascentes que procuram resolver o problema, como painéis solares incorporados na tela ou dispositivos cinéticos que aproveitam seus movimentos para carregar um aparelho. Mas estão a um longo caminho de oferecer uma solução prática, pois praticamente seria necessário um novo tipo de bateria.

Com base nisso, podemos assumir que a bateria é um recurso restritivo. As principais estratégias para melhorar o seu tempo de utilização em dispositivos móveis ainda são: baterias extras e aplicações personalizadas para preservar a fonte de energia. Uma vez que as baterias têm um número máximo de ciclos de vida, a descarga prematura de cada ciclo não apenas impacta sobre a disponibilidade do dispositivo móvel em um curto período de tempo, como também afeta a sua capacidade para funcionar durante um longo período de tempo, ou seja, a velocidade de descarga impacta na vida útil da bateria.

Estes problemas com a computação móvel podem prejudicar algumas aplicações que demandam mais recursos para atender às necessidades e expectativas dos utilizadores (SATYANARAYANAN et al., 2009). Nesse sentido, a *Mobile Cloud Computing* (MCC) visa superar as

[1]Uma aplicação é considerada em produção quando esta já passou por todas as fases de desenvolvimento (incluindo testes), foi implantada e já está sendo utilizadas por seus usuários.

limitações da computação móvel, permitindo assim a entrega de aplicações mais sofisticadas e inovadoras através do uso de recursos fornecidos em nuvem. No entanto, ao implantar uma infraestrutura de MCC, o provedor do serviço não possui garantias de que o serviço será provido de maneira adequada. Por isso, há a necessidade de efetuar um planejamento adequado, levando em considerações questões que envolver a qualidade so serviço prestado, como a disponibilidade e a confiabilidade.

Nesse contexto, esta tese propõe um conjunto de modelos para auxiliar no planejamento de infraestruturas de *mobile cloud computing*. A avaliação é baseada em modelos heterogêneos hierárquicos e se concentra em métricas como a disponibilidade em estado estacionário, confiabilidade, *downtime*, probabilidade de conectividade, tempo médio de vida da bateria, e os custos de implantação e provisionamento da infraestrutura de nuvem. Propomos uma forma de apoiar a escolha entre nuvens privadas e públicas, comparando cenários diferentes para avaliação de fatores como a aquisição de *hardware*, consumo de energia, impacto relativo à conectividade de rede (locais e móveis), dos protocolos de comunicação, do tempo de vida das baterias e do tempo médio de entrega de mensagens entre dispositivos móveis.

1.3 Objetivos

O principal objetivo desta tese é propor um conjunto de modelos estocásticos para planejamento de infraestruturas de *mobile cloud computing*, com base na modelagem hierárquica e heterogênea dessa infraestrutura. Esta abordagem deve permitir a identificação de pontos de melhoria do sistema. Os modelos propostos permitirão identificar qual infraestrutura de sistema é capaz de prover serviços de *mobile cloud computing* que atendam requisitos desejáveis. Os objetivos específicos da pesquisa são:

- Propor modelos estocásticos para planejamento de infraestruturas de *mobile cloud computing*, considerando principalmente falhas e reparos de *hardware* e *software*;
- Identificar o impacto de diferentes protocolos de comunicação e cenários de conectividade no consumo de energia e *downtime* de dispositivos móveis;
- Identificar o tempo médio de entrega de mensagens em uma infraestrutura de *mobile cloud computing* considerando diferentes tipos de rede;
- Propor melhorias arquiteturais e ajustes em parâmetros de infraestrutras de MCC através dos modelos propostos e dos resultados de análise de sensibilidade.

1.4 Principais contribuições

As principais contribuições que apresentamos nesta tese são:

- Modelos para avaliação da disponibilidade e confiabilidade de ambientes de *mobile cloud computing*, abordando problemas de infraestrutura e de aplicação;
- Identificação dos parâmetros mais importantes para a disponibilidade e confiabilidade da infraestrutura através de análise de sensibilidade;
- Modelos estocásticos para avaliação da conectividade da rede e do tempo de descarga e tempo de vida da bateria;
- Identificação do impacto de diferentes canais de comunicação no tempo de descarga da bateria de dispositivos móveis;
- Modelos de entrega de mensagens para sistemas de MCC que identifica o tempo médio de entrega de mensagens através de diferentes tipos de redes de comunicação;
- Método e ferramental para automatização do monitoramento de recursos distribuídos em nuvens privadas;
- Ferramental de alto nível para predição de utilização de recursos computacionais;
- Modelos de custo implantação e provisionamento de nuvens privadas e públicas para sistemas de *mobile cloud computing*;
- Uma metodologia de apoio que descreve as atividades necessárias para a adequada definição e análise de modelos.

1.5 Organização do documento

Esta de tese está estruturada da seguinte forma. O Capítulo 2 apresenta conceitos básicos de temas relevantes para que o leitor possa entender corretamente este documento. O Capítulo 3 discute pesquisas relevantes encontradas na literatura que têm alguns temas em comum para aqueles abordados nesta tese. A metodologia de apoio adotada nessa tese está descrita no Capítulo 4. Já o Capítulo 5 explica os modelos propostos juntamente com as respectivas arquiteturas abordadas. O Capítulo 6 apresenta os estudos de casos que foram utilizados para avaliar os cenários de interesse, bem como os resultados alcançados durante o doutorado. As conclusões e trabalhos futuros são discutidos no Capítulo 7.

2

Fundamentação teórica

As oportunidades multiplicam-se na medida em que são agarradas.

—SUN TZU - A ARTE DA GUERRA

Este capítulo apresenta os conceitos básicos das principais áreas abordadas nesta tese: *mobile cloud computing*, dependabilidade e modelagem estocástica, análise de sensibilidade e envelhecimento e rejuvenescumento de *software*. O referencial teórico aqui apresentado deve fornecer o conhecimento necessário para uma compreensão clara dos capítulos à frente.

2.1 *Mobile cloud computing*

Com o crescimento acelerado da computação móvel e a popularização das tecnologias de rede sem fio, dispositivos inteligentes (*smartphones* e *tablets*, por exemplo) estão se tornando uma parte essencial de nossas atividades diárias. Como resultado, essas tecnologias estão trazendo benefícios às diversas áreas da vida humana, como o comércio, *e-learning*, saúde, entre outros.

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 a seus usuários permanecerem com tais dispositivos a maior parte do tempo (DISTERER; KLEINER, 2013). A segunda propriedade, importante para a computação móvel, é a conectividade, isto é, a capacidade de receber e enviar dados para a Internet através das 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 princialmente no processamento e na bateria, além de por diversas vezes não poder contar com uma conexão de rede adequada.

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 (bateria e conhexão de rede, por exemplo), como do lado servidor (falhas de *hardware* e *software*, por exemplo). Para tentar minimizar esses problemas e as limitações dos dispositivos, a computação móvel e a computação em nuvem, 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). A Figura 2.1 apresenta a arquitetura de uma *mobile cloud*.

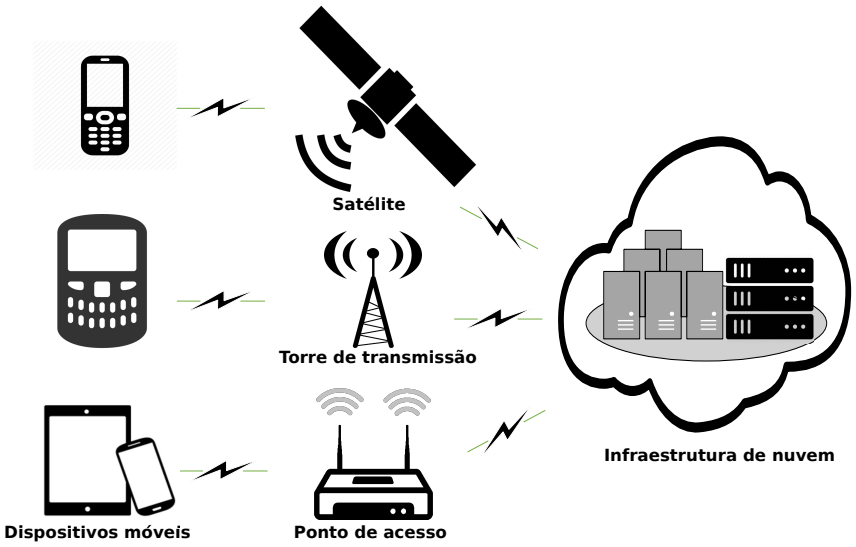


Figura 2.1: Representação da arquitetura de uma *mobile cloud*

O paradigma da *mobile cloud computing* surgiu nos últimos anos como a intersecção de dois campos importantes de TI: computação móvel e computação em nuvem (DINH et al., 2013). Como o mercado de computação móvel cresce, os usuários exigem recursos mais avançados em aplicações móveis, exigindo o mesmo nível de complexidade das aplicações *desktop*. No entanto, até mesmo os *smartphones* mais modernos sofrem pela limitação de recursos computacionais como CPU, memória, armazenamento e bateria. Embora tais dispositivos possuam recursos limitados, infraestruturas de computação em nuvem podem proporcionar poder computacional sob demanda. Portanto, a *mobile cloud computing* pode ser definida como uma extensão da computação móvel, onde a computação e armazenamento intensivo são movidos dos dispositivos móveis para a nuvem (MCC Forum, 2013). O encontro entre a computação móvel e a computação em nuvem é possível devido ao número crescente de usuários móveis com acesso à Internet de banda larga.

O principal objetivo da MCC é usar a capacidade de processamento e armazenamento de uma nuvem computacional para estender as capacidades de dispositivos móveis com poder computacional 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).

Uma arquitetura típica de computação em nuvem é constituída por três modelos de

serviço principais: *Software as a Service* (SaaS), *Platform as a Service* (PaaS) e *Infrastructure as a Service* (IaaS) (MELL; GRANCE, 2011). O modelo de *software* como um serviço (SaaS) proporciona ao usuário usar uma determinada aplicação disponível através da Internet e acessível via navegador ou por uma aplicação cliente em qualquer dispositivo. Em SaaS, o usuário não gerencia nem controla a infraestrutura subjacente da nuvem, apenas pode definir as configurações dos próprios aplicativos; a plataforma como um serviço (PaaS) provê um ambiente onde os desenvolvedores podem usar bibliotecas, serviços e ferramentas disponibilizadas pelos provedores para criar, implantar e gerenciar aplicações. O usuário (desenvolvedor) de PaaS não gerencia nem controla a infraestrutura subjacente na nuvem, porém tem controle sobre os aplicativos implantados e possivelmente define as configurações no ambiente de hospedagem das aplicações; já a infraestrutura como um serviço (IaaS) é responsável por fornecer processamento, armazenamento, rede e outros recursos computacionais fundamentais que possibilitem o usuário implantar e executar um *software* arbitrário, incluindo sistemas operacionais e aplicações. O usuário não pode gerir ou controlar a infraestrutura subjacente da nuvem, mas tem o controle sobre os sistemas operacionais, armazenamento e aplicativos implantados; e possivelmente tem o controle, mesmo que limitado, de componentes de rede, como *firewalls*.

Além disso, a computação em nuvem é formada por quatro modelos de implantação (MELL; GRANCE, 2011):

- **Nuvem privada**: A infraestrutura de nuvem é implantada, localmente ou não, para uso exclusivo por uma única organização, podendo ser gerenciada pela própria organização, por terceiros ou ambos;
- **Nuvem comunitária**: A infraestrutura de nuvem é implantada para uso exclusivo de um conjunto de organizações que compartilham dos mesmos interesses. O gerenciamento dessa nuvem pode ficar por conta de algumas dessas empresas ou por terceiros;
- **Nuvem pública**: A infraestrutura de nuvem é implantada nas instalações do provedor e o serviço por ela oferecido é aberto e voltado para o público geral. Este tipo de implantação pode ser gerenciada por empresas, instituições de ensino, organizações governamentais ou uma combinação destas;
- **Nuvem híbrida**: Neste modelo de implantação, a infraestrutura de nuvem é composta de dois ou mais infraestruturas distintas que permanecem como uma entidade única, mas são ligadas por uma tecnologia padronizada ou proprietária que permita a portabilidade de dados e aplicações.

2.2 Dependabilidade

Há muitos requisitos funcionais e não-funcionais que um aplicativo deve cumprir para garantir a satisfação e fidelidade dos usuários. A dependabilidade aborda alguns desses principais requisitos.

O conceito de **dependabilidade** está relacionado com **dependência** e **confiança**. Se um sistema *A* depende de um sistema *B*, então falhas em *B* podem afetar *A*. A **confiança** é, portanto, "a **dependência aceita** entre os sistemas que estão sendo avaliados". Assim, a **dependabilidade** é "a capacidade de oferecer serviços que podem ser confiáveis, respeitando um nível aceitável de frequência e gravidade das falhas" (AVIZIENIS et al., 2004). Portanto, a ocorrência de ameaças à dependabilidade deve ser evitada na medida do possível e atenuada sempre que elas ocorrem.

A dependabilidade é frequentemente dividida em atributos quantitativos e qualitativos como disponibilidade, confiabilidade, segurança, integridade e manutenabilidade (AVIZIENIS et al., 2004):

- **Disponibilidade**: capacidade de fornecer serviços de forma instantânea;
- **Confiabilidade**: capacidade de fornecer serviços de forma contínua;
- **Segurança**: ausência de consequências nocivas aos usuários e ao ambiente;
- **Integridade**: ausência de alterações indesejadas nos sistemas;
- **Manutenabilidade**: capacidade de realizar mudanças de sistemas e reparos.

Além dos atributos descritos, outros também podem estar presentes no estudo de dependabilidade de sistemas, tais como **confidencialidade**, que é a capacidade do sistema evitar o acesso não autorizado à informação (AVIZIENIS et al., 2004; MACIEL et al., 2012). A **disponibilidade**, **confiabilidade** e a **manutenabilidade** são atributos quantitativos, enquanto a **segurança** e a **integridade** são atributos qualitativos. Uma vez que estamos preocupados com uma avaliação quantitativa e para o propósito desse estudo, avaliamos a dependabilidade dos sistemas com foco na **disponibilidade** e **confiabilidade**. A Figura 2.2 apresenta a relação entre dependabilidade e os atributos, ameaças e meios.

No estudo da dependabilidade dos sistema, as ameaças à disponibilidade são *faults, errors* e *failures* (AVIZIENIS et al., 2004; MACIEL et al., 2012). Uma *fault* é "um funcionamento anormal de um componente do sistema, subsistema ou outro sistema que o sistema a que se refere depende". Assim, uma *fault* pode causar um *error* "no estado do serviço de um componente que é uma parte do estado interno do sistema". Se uma *fault* provoca um *error*, então a *fault* é **ativa**, caso contrário é **dormente**. Às vezes, um *error* não tem consequências sobre o estado externo de um sistema, mesmo afetando o seu estado interno. Uma *failure* é "a incapacidade do sistema de entregar os serviços especificados, quando um erro impacta seu estado externo". Um **sistema tolerante a falhas** é um sistema que "continua a sua operação pretendida mesmo que uma ou

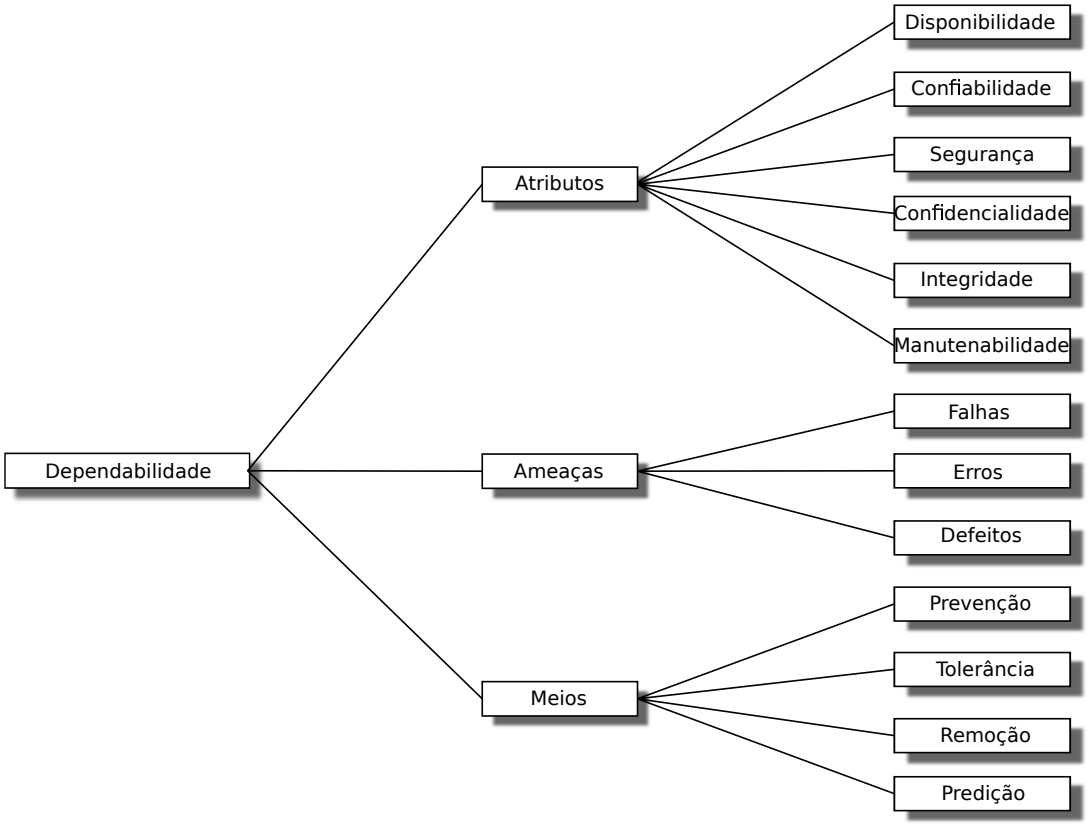


Figura 2.2: Relação entre dependabilidade e os atributos, ameaças e meios (traduzido de (AVIZIENIS et al., 2004)).

mais falhas ocorram". Assim, um sistema tolerante a falhas é concebido para suportar tais falhas e continuar ativo, usando mecanismos como a redundância de módulos, que evita o seu fracasso completo.

A taxa de falhas é modelada usando uma curva da banheira, como mostrado na Figura 2.3. Esta curva é dividida em três regiões, que representam cada fase do ciclo de vida de um produto ou serviço. Na fase **mortalidade infantil**, o produto ou serviço é iniciado com um elevado grau de falhas. Na fase de **desgaste**, o produto ou serviço está se tornando obsoleto e, assim, aumenta a sua taxa de falha. Neste trabalho, estamos interessados na região do **período de vida útil**, onde o produto tem uma taxa de falha constante.

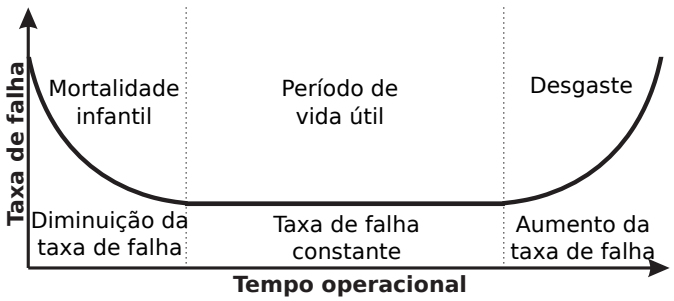


Figura 2.3: Modelo de curva da banheira.

Em geral, a confiabilidade dos sistemas pode ser calculada a partir da Equação 2.1.

$$R(t) = e^{-\int \lambda(t)dt}. \tag{2.1}$$

onde e representa uma distribuição exponencial; $\lambda(t)$ é a taxa em função do tempo; e dt é a derivada do tempo.

Para o caso particular em que a taxa de falha é constante, a confiabilidade é reduzida a $R(t) = e^{-\lambda t}$ (MACIEL et al., 2012; TRIVEDI, 2002).

Embora a função de confiabilidade $R(t)$ forneça "a probabilidade do sistema funcionar muito bem até ou no tempo t", a função $F(t)$ (ver Equação 2.2) é "a probabilidade de ocorrer pelo menos uma falha até ou no tempo t". $F(t)$ é uma **função de probabilidade cumulativa** de falhas ao longo do tempo. A **função densidade de probabilidade** de falhas em um instante t ($f(t)$) é, portanto, a derivada de $F(t)$, como representado na Equação 2.3.

$$F(t) = 1 - R(t) \tag{2.2}$$

$$f(t) = \frac{dF(t)}{dt} = -\frac{dR(t)}{dt} \tag{2.3}$$

O estado estacionário da disponibilidade A é a probabilidade do sistema estar **trabalhando** (mesmo que não seja em sua capacidade total) ao longo do tempo. Assim, a disponibilidade é calculada usando a Equação 2.4, tendo ***Up Time*** como o tempo total que o sistema está sendo executado, e o ***Down Time*** como o tempo total que o sistema não está em execução. Sempre que não for possível avaliar estes valores, a disponibilidade pode ser avaliada a partir das falhas e reparos do sistema.

$$A = \frac{Up\ Time}{Up\ Time + Down\ Time} \tag{2.4}$$

Considerando que o sistema está no estado estacionário, o *Mean Time Between Failures* (MTBF) é aproximadamente igual ao *Mean Time to Failure* (MTTF). Por sua vez, o *Mean Time to Repair* (MTTR) é o tempo médio necessário para reparar uma falha. Portanto, a disponibilidade em estado estacionário A pode ser avaliada utilizando a Equação 2.5. O $MTTF$ pode ser calculado a partir da Equação 2.6. Já o $MTTR$ pode ser calculado a partir da Equação 2.7 (MACIEL et al., 2012).

$$A = \frac{MTTF}{MTTF + MTTR} \tag{2.5}$$

$$MTTF \cong MTBF = \int_0^\infty R(t) \times dt, \tag{2.6}$$

$$MTTR = \int_0^\infty M(t) \times dt, \tag{2.7}$$

onde M refere-se à manutenabilidade, ou seja, à capacidade de fazer reparos no sistema.

Por vezes, a disponibilidade tem percentagens com alta ordem de magnitude. Tais magnitudes são muitas vezes referidos pela **número de noves** nos dígitos. Por exemplo, se um sistema dispõe de uma disponibilidade de 99,999%, é possível dizer que a sua disponibilidade é de 5 noves. Esta notação é a preferida para valores de magnitude elevada, uma vez que é mais legível. Seja x um valor percentual, ele pode ser convertido para o número da notação de noves usando a Equação 2.8. Esta equação utiliza as percentagens expressas no intervalo de zero a um, assim, 99,999% equivale a 0,99999.

$$N = 2 - log(100 - A) \tag{2.8}$$

Alcançar níveis satisfatórios de dependabilidade pode ser um desafio em diversos sistemas. O custo do sistema aumenta exponencialmente com a dependabilidade. Assim, os sistemas altamente confiáveis tendem a ser muito caros. É o caso dos sistemas de MCC, que precisa de vários dispositivos alimentados por bateria conectados através de redes sem fios e móveis, torna-se ainda mais difícil de alcançar altos níveis de dependabilidade. Por exemplo, uma descarga de bateria e o bloqueio de sinal afetam a disponibilidade e a confiabilidade do sistema. Além disso, também podemos listar o envelhecimento de *software* como sendo um dos fatores mais importantes que afetam a dependabilidade dos sistemas.

2.3 Modelagem estocástica

Modelagem estocástica refere-se ao uso da probabilidade para modelar situações do mundo real em que a incerteza está presente. É uma técnica aplicável a quaisquer processos que possam ser medidos estatisticamente. Um processo estocástico é uma variável que se comporta, durante o tempo, de uma maneira onde pelo menos parte é considerada randômica.

O uso de um modelo estocástico não implica que o modelador acredite fundamentalmente que o sistema em questão se comporta “aleatoriamente”. (Por exemplo, o comportamento de um indivíduo pode parecer “aleatório”, mas uma entrevista com esse indivíduo pode revelar um conjunto de preferências sob as quais o comportamento dessa pessoa é revelado como totalmente previsível.) O uso de um modelo estocástico reflete apenas uma decisão pragmática por parte do modelador, que tal modelo representa a melhor descrição atualmente disponível do fenômeno considerado, considerando os dados disponíveis e o universo de modelos conhecidos pelo modelador.

Modelagem estocástica é uma abordagem importante para realizar estudos de dependabilidade. Modelos estocásticos para modelagem de disponibilidade e confiabilidade podem ser classificados em duas categorias: modelos combinatórios e baseados no estado (MACIEL et al., 2012). Modelos combinatórios representam o modo de falha ou o modo operacional por relações estruturais entre os componentes e subsistemas que compõem o sistema. Exemplos de modelos combinatórios são os diagramas de bloco de confiabilidade (do inglês, *Reliability Block Diagram* (RBD)) e as *fault trees* (O’CONNOR; O’CONNOR; KLEYNER, 2012). Modelos

baseados em estado descrevem o comportamento do sistema em termos de estados e ocorrências de eventos expressados como as transições de estado marcados. Cadeias de Markov e redes de Petri estocásticas (do inglês, *Stochastic Petri Nets* (SPN)) são dois tipos principais de modelos baseados no estado.

Há alguns *tradeoffs*[1] entre essas duas categorias de modelos (MALHOTRA, 1994). Modelos combinatórios são mais simples e mais fáceis de criar, mas, muitas vezes, tem de assumir que os componentes são estatisticamente independentes. A suposição de independência pode ser demasiada restritiva, portanto, limitando o poder expressivo dessa classe de modelos. Modelos baseados no estado são mais complexos e difíceis de criar, mas eles possuem maior expressividade do que os modelos combinatórios, e calculam métricas. Não raramente, as métricas de modelos baseados no estado devem ser avaliadas por meio de simulação, devido ao problema de explosão de espaço de estados. Com modelos baseados no estado, podemos representar um comportamento mais complexo, como os esquemas de redundância TMR (do inglês, *Triple Modular Redundancy*), *hot-standby*, *warm-standby*, e *cold-standby* (LOGOTHETIS; TRIVEDI, 1995).

A combinação dos dois tipos de modelos também é possível, permitindo assim a obtenção do melhor dos dois mundos, por meio da modelagem heterogênea (KIM; GHOSH; TRIVEDI, 2010; MALHOTRA; TRIVEDI, 1993). Modelos distintos, do mesmo tipo ou de tipos diferentes, podem ser dispostos em diferentes níveis de compreensão, levando a modelos hierárquicos (MALHOTRA; TRIVEDI, 1993).

Normalmente, um modelo de nível superior representa o sistema baseado em seus subsistemas (ou seja, grupos de componentes relacionados), enquanto os modelos de nível inferior descrevem o comportamento detalhado de cada subsistema. Modelos heterogêneos hierárquicos têm sido utilizados para lidar com a complexidade de muitos tipos de sistemas, por exemplo, redes de sensores (KIM; GHOSH; TRIVEDI, 2010), telecomunicações (TRIVEDI et al., 2006), nuvens privadas (DANTAS et al., 2012), e sistemas de *mHealth* (ARAUJO et al., 2014).

Nesta tese adotamos somente os modelos RBD e cadeias de Markov, razão pela qual iremos detalhar um pouco mais sobre esses dois formalismos.

2.3.1 Diagramas de bloco de confiabilidade

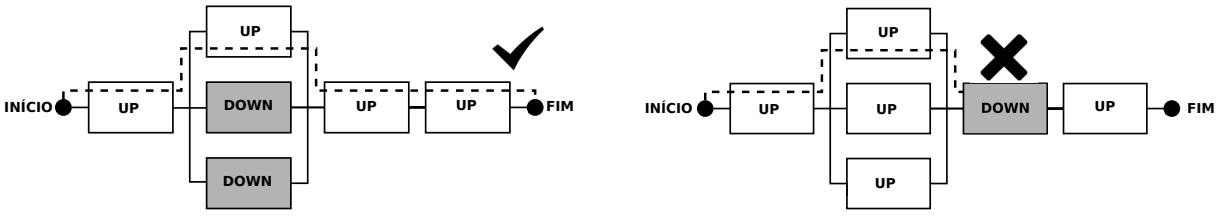
Um diagrama de bloco de confiabilidade – do inglês, *Reliability Block Diagram* (RBD) – é um modelo combinatório que é comumente usado para modelagem de dependabilidade. Nestes modelos, a falha e a recuperação de um componente são independentes de outros componentes. Assim, o comportamento de um componente não pode afetar a taxa de insucesso do outro.

RBDs modelam sistemas com base em seus componentes, muitas vezes representado como retângulos e as conexões entre eles. O modelo tem uma única fonte, um único alvo, e

[1] É uma expressão que define uma situação em que há conflito de escolha.

seus componentes que estão conectados usando arcos não-direcionais. Os componentes são geralmente ligados em série ou em paralelo, embora haja outros tipos de estruturas especiais (por exemplo, *K-out-of-N*, *brige*, etc.) (MACIEL et al., 2012).

Para cada bloco, é necessário atribuir a probabilidade do componente estar em modo operacional e de falha. O $MTTF$ e o $MTTR$ são valores que podem ser utilizados para este fim, de modo a calcular a confiabilidade e a disponibilidade. Considerando que cada bloco representa um componente do sistema, o sistema irá falhar se não houver um caminho disponível do início ao fim. Isso é ilustrado na Figura 2.4. A Figura 2.4(a) mostra que o sistema permanece disponível apesar da falha de dois componentes, pois temos um caminho contíguo desde a origem até o vértice final. Na Figura 2.4(b), a falha de um componente é crítica para o sistema, uma vez que não existe um caminho no RBD da origem até o final.



(a) Sistema disponível. (b) Sistema indisponível.

Figura 2.4: Exemplo de disponibilidade usando RBD.

Nos blocos em série, se um único componente falha, todo o sistema não está operacional. Considerando um RBD com **todos os componentes em série**, sendo $R_i(t)$ a confiabilidade de um componente i no tempo t e sendo $R_s(t)$ a confiabilidade do sistema em série, com n componentes, em tempo de t é:

$$R_s(t) = \prod_{i=1}^{n} R_i(t) \tag{2.9}$$

Do mesmo modo, a disponibilidade instantânea do sistema é:

$$A_s(t) = \prod_{i=1}^{n} A_i(t) \tag{2.10}$$

E a disponibilidade do estado estacionário é:

$$A_s = \prod_{i=1}^{n} A_i \tag{2.11}$$

onde A_i é a disponibilidade do estado estacionário do i-ésimo componente.

Considerando um RBD com **todos os componentes em paralelo**, sendo $R_i(t)$ a confiabilidade de um componente i no tempo t, e sendo $R_p(t)$ a confiabilidade de um sistema, com n componentes, em tempo de t é:

$$R_p(t) = 1 - \prod_{i=1}^{n}(1 - R_i(t)) \tag{2.12}$$

Do mesmo modo, a disponibilidade instantânea do sistema é calculada através da Equação 2.13.

$$A_p(t) = 1 - \prod_{i=1}^{n}(1 - A_i(t)) \tag{2.13}$$

E a disponibilidade do estado estacionário é expressada pela Equação 2.14.

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

onde A_i é a disponibilidade do estado estacionário do i-ésimo componente.

Um sistema RBD pode ser decomposto em vários subsistemas, e as equações descritas podem ser usadas sobre cada subsistema. Os resultados dos subsistemas podem então ser usados para calcular a probabilidade de todo o sistema.

2.3.2 Cadeias de Markov

Modelos de Markov são os blocos de construção fundamentais sobre os quais a maioria das técnicas quantitativas de análise de desempenho são construídas (KOLMOGOROV, 1931; TRIVEDI, 2002). Tais modelos podem ser utilizados para representar as interações entre os vários componentes do sistema, tanto para fins descritivos, quanto para fins preditivos. Os modelos de Markov têm sido utilizados intensivamente na modelagem de dependabilidade e de desempenho desde por volta dos anos cinquenta (MACIEL et al., 2012). Além da ciência da computação, a gama de aplicações para os modelos de Markov é muito extensa. Economia, meteorologia, física, química e telecomunicações são alguns exemplos de campos que encontraram neste tipo de modelagem estocástica[2] uma boa abordagem para resolver vários problemas.

Sendo $Pr\{k\}$ a probabilidade de um determinado evento k ocorrer. Um processo de Markov é um processo estocástico em que $Pr\{X_{t_{n+1}} \leq s_{i+1}\}$ depende apenas do último valor X_{t_n}, para todo $t_{n+1} > t_n > t_{n-1} > ... > t_0 = 0$, e todo $s_i \in S$. Esta é a chamada propriedade de Markov (HAVERKORT, 2002), que, em palavras simples, significa que a evolução futura do processo de Markov é totalmente descrito pelo estado atual, e é independente de estados passados (HAVERKORT, 2002).

Os modelos de Markov de espaço de estado discretos (contáveis), também conhecidos como cadeias de Markov, se distinguem em duas classes: **cadeia de Markov de tempo discreto** (do inglês, *Discrete Time Markov Chain* (DTMC)), e **cadeia de Markov de tempo contínuo** (do inglês, *Continuous Time Markov Chain* (CTMC)). Cadeias de Markov são modelos de espaço de

[2]Estocástico se refere a algo que envolve ou contém uma variável ou variáveis aleatórias.

estados (ou modelos não-combinatórios). Ao contrário do RBD, elas podem modelar relações complexas entre os componentes do sistema, incluindo alterações em falhas e taxas de reparo com base em no comportamento de outros componentes.

Embora as cadeias de Markov sejam capazes de representar relações mais complexas do que RBDs, eles também têm um custo computacional superior. Assim, dependendo do tamanho do modelo, pode ser muito difícil ou mesmo impossível de calcular as métricas. No entanto, assim como no RBD, estes modelos podem ser convertidos em fórmulas fechadas que são utilizadas para avaliar as métricas e podem ser utilizadas para uma análise de sensibilidade paramétrica.

A Figura 2.5 mostra uma cadeia de Markov de tempo contínuo. Nesse exemplo, o sistema pode ir do estado S_0 para S_1 através da transição q_{01}. A partir daí, pode retornar para o estada S_0 através da transição q_{10}, ou ir para o estado S_2 através da transição q_{12}. Agora no estado S_2, pode retornar para S_1 a partir da transição q_{21}.

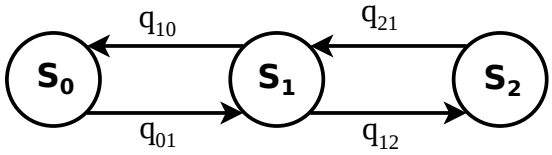


Figura 2.5: Exemplo de cadeia de Markov.

Sendo S o conjunto de estados de uma CTMC, a probabilidade do estado estacionário estar em cada estado $i \in S$ é calculada resolvendo o sistema de equações lineares:

$$\begin{cases} \pi Q = 0 \\ \sum \pi_i = 1, \quad \forall i \in S \end{cases} \tag{2.15}$$

onde π é o vetor de probabilidades do estado, Q é a uma matriz com as taxas de transição de modo que q_{ij} é a taxa de transição do estado i a j. Os elementos da diagonal principal de Q são definidos como $q_{ii} = -\sum_{j,j\neq i} q_{ij}$.

Além disso, nesse trabalho também utilizamos um outro tipo de modelo, chamado de **Modelo de estado absorvente**. Vamos apresentar algumas definições que vão ajudar a entender esse conceito.

Os estados de uma CTMC ou DTMC podem ser classificados de acordo com a alcançabilidade através de outros estados. A CTMC como um todo também é classificada de acordo com seus estados. Se há um caminho a partir de um estado i para um estado j, então esses dois estados podem se *comunicar*. A notação empregada para designar esse fato é: $i \leftrightharpoons j$. Com base nesta notação, vamos definir o conceito de *classe*:

Definição 2.3.1. *A classe $C[i]$ de um estado i é o conjunto de todos os estados j que comunica com i, ou seja, $C[i] = \{j | i \leftrightharpoons j; j \in S\}$*

onde S é o espaço estado da cadeia de Markov.

Com a ajuda da definição de classe, agora vamos apresentar a definição de classes ergódicas e transitórias:

Definição 2.3.2. *Uma classe C tal que cada caminho que começa em C também termina em C, é classificado como uma classe* ergódica*.*

Definição 2.3.3. *Uma classe C tal que existe um caminho que começa em C e termina em uma classe diferente, é classificado como uma classe* transiente*.*

Os estados que pertencem a uma classe transiente também são definidos como *estados transientes*.

Se uma classe ergódica consiste em apenas um estado, esse estado é dito ser *absorvente*. Isto significa que este estado não se comunica com qualquer outro estado. Uma vez que a cadeia de Markov atinge este estado, ele vai ficar lá para sempre. Finalmente, definimos o conceito de uma cadeia de Markov absorvente:

Definição 2.3.4. *Se uma cadeia de Markov contém apenas estados absorventes ou transientes, é definido como uma* cadeia de Markov absorvente*.*

Para cadeias de Markov de tempo discreto, as métricas de interesse de uma cadeia de absorção são os seguintes: i) a probabilidade de se chegar a um determinado estado absorvendo a partir de um estado transiente; ii) o número de vezes que qualquer estado transiente é visitado antes que a cadeia chega a um estado de absorção, a partir de um estado transiente específico (WINSTON; GOLDBERG, 1994).

Os leitores podem obter uma definição completa sobre cadeias de Markov em (TRIVEDI, 2002; BOLCH et al., 1998).

2.4 Análise de sensibilidade

A análise de sensibilidade é uma técnica utilizada para determinar os fatores que possuem maior relevância sobre as medidas ou saídas de um modelo. Em análise de dependabilidade de sistemas computacionais, é possível aplicar a análise de sensibilidade para auxiliar na identificação dos componentes que mais influenciam para a disponibilidade, confiabilidade ou desempenho de um sistema (HAMBY, 1994).

De acordo com HAMBY (1994), existe um consenso entre autores que os modelos são de fato sensíveis a duas formas distintas de parâmetros de entrada: a variabilidade, ou incerteza, associada a um parâmetro de entrada sensível tem uma grande contribuição à variabilidade da saída global; e pode haver uma alta correlação entre um parâmetro de entrada e os resultados do modelo, ou seja, pequenas mudanças nos valores de entrada resultam em mudanças significativas na saída.

Uma análise de sensibilidade pode ser feita por meio de diferentes métodos, entre eles estão: análise diferencial, análise de correlação, análise de regressão, análise de perturbação e

DoE (HAMBY, 1994; MATOS et al., 2015; BEZERRA et al., 2014). O método mais simples é variar repetidamente um parâmetro de cada vez, mantendo os outros constante. Ao aplicar este método, um *ranking* de sensibilidade é obtido observando as alterações da saída do modelo. Este método é comumente utilizado em conjunto plotando gráficos de entrada *versus* a saída. Tais gráficos permitem a detecção gráfica de não-linearidades, não monotonicidade, e correlações entre as entradas e saídas do modelo (MARINO et al., 2008).

A análise de sensibilidade diferencial é realizada calculando as derivadas parciais da medida de interesse com relação a cada parâmetro de entrada. Assim, a sensibilidade de uma determinada medida Y, que depende de um parâmetro específico θ, é calculado a partir da Equação 2.16, ou (2.17) para uma sensibilidade dimensionada.

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

$$SS_\theta(Y) = \frac{\lambda}{Y}\frac{\partial Y}{\partial \theta}. \tag{2.17}$$

onde, $S_\theta(Y)$ é o índice (ou coeficiente) de sensibilidade de Y em relação a θ, e $SS_\theta(Y)$ é o índice de sensibilidade dimensionado, comumente usado para contrabalançar os efeitos de diferentes unidades entre os valores de parâmetros distintos.

Métodos específicos para realizar a análise de sensibilidade diferencial em modelos analíticos são necessários quando não há equações de fórmulas fechada diretas para o cálculo da medida de interesse, e encontrar sua expressão derivada. Muitos trabalhos já descreveram como aplicar análise de sensibilidade diferencial em uma variedade de modelos analíticos, incluindo CTMC (BLAKE; REIBMAN; TRIVEDI, 1988), MRM (ABDALLAH; HAMZA, 2002), GSPN (MUPPALA; TRIVEDI, 1990). Ao lidar com modelos hierárquicos ou compostos, a análise de sensibilidade deve considerar todos os parâmetros de cada modelo, determinando o seu impacto à medida global de interesse. Uma equação de fórmula fechada com base em medidas de cada sub-modelo pode ser usada para esses casos, de modo que as derivadas parciais calculadas para os sub-modelos nos permite obter um *ranking* de sensibilidade completo.

Derivadas parciais são um meio importante de executar análise de sensibilidade, mas elas podem não avaliar adequadamente a sensibilidade com relação a parâmetros com valores inteiros, porque a abordagem é projetada para valores de entrada de parâmetro em um domínio contínuo. Uma abordagem para resolver essa questão é baseado no cálculo da diferença percentual ao variar um parâmetro de entrada a partir do seu valor mínimo para seu valor máximo. Hoffman e Gardner (HOFFMAN; GARDNER, 1983) defendem a utilização de cada parâmetro de valores possíveis para calcular as sensibilidades dos parâmetros.

Nesta tese adotamos métricas de importância dos componentes. Tais métricas são usadas para avaliar o impacto de cada componente sobre a disponibilidade ou confiabilidade global do sistema. Com a ajuda destas métricas, podemos classificar os componentes do sistema e identificar quais os que promovem maiores impactos sobre a dependabilidade do sistema. Desta

forma, a fim de obter melhorias em termos de dependabilidade geral do sistema, podemos nos concentrar na melhoria desses componentes, substituindo-os por componentes mais confiáveis, ou a adição de mecanismos de redundância.

Existem várias métricas de importância dos componentes na literatura. Uma das mais utilizadas é a *reliability importance*, também conhecida como *Birnbaum importance* (*B-importance*) (BIRNBAUM, 1968). Esta métrica é definida como o aumento na disponibilidade quando a confiabilidade dos componentes é aumentada em uma unidade. Existem várias maneiras de expressar essa métrica. Nesta seção, iremos mostrar duas formas. A primeira é expressar a *B-importance* como a derivada parcial da confiabilidade do sistema em relação com a confiabilidade de um componente *i* (Equação 2.18).

$$I^B(i|t) = \frac{\partial R_{sys}(t)}{\partial p_i(t)}, \tag{2.18}$$

onde, I_i^B é o índice *B-importance* para o componente *i*; p_i é a confiabilidade do componente *i*; R_{sys} é a confiabilidade do sistema, e t é o tempo considerado para medir a confiabilidade.

Com base na definição anterior, e considerando que $0 \leq p_i \leq 1$, a fórmula para a obtenção do *B-importance* pode ser expressa como:

$$I^B(i|t) = R_s(1_i, \boldsymbol{p}^i) - R_s(0_i, \boldsymbol{p}^i) \tag{2.19}$$

onde, p^i representa o vetor de confiabilidade do componente do sistema com o componente *i* removido; 0_i representa a condição de quando o componente *i* falhou; e 1_i representa a condição de quando o componente *i* estiver no estado operacional. Em resumo, a importância da confiabilidade é simplesmente a diferença entre a confiabilidade do sistema, quando o componente *i* está funcionando, e a confiabilidade quando o componente *i* falhou (RAUSAND; HØYLAND, 2004), conforme mostrado na Figura 2.6.

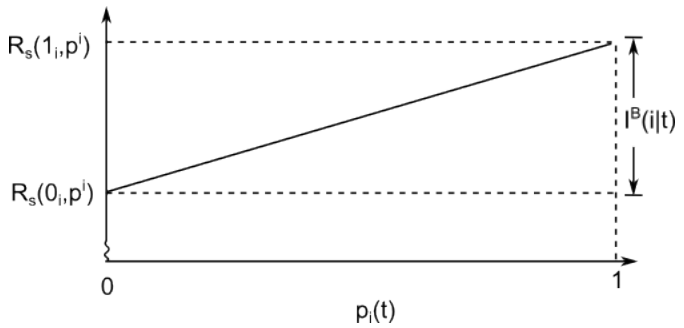


Figura 2.6: Visualização gráfica da métrica *B-importance* (RAUSAND; HØYLAND, 2004)

De forma semelhante, definimos a métrica *availability importance* como (BARABADY; KUMAR, 2007):

$$I_i^A = \frac{\partial A_s}{\partial A_i} \tag{2.20}$$

onde, A_s é a disponibilidade do sistema, e A_i é a disponibilidade do componente. O valor obtido a partir desta fórmula é interpretado da mesma maneira do *B-importance*. É um valor que varia de 0 a 1, sendo que quanto maior o valor, maior é o impacto do componente sobre a disponibilidade do sistema.

2.5 Envelhecimento e rejuvenescimento de *software*

Uma das fases mais importantes no desenvolvimento de *software* é a fase de testes. Geralmente, nesta fase, vários testes são realizados para verificar o comportamento do sistema sob diferentes condições. O principal objetivo desta fase é encontrar possíveis erros ou *bugs* no *software* e corrigi-los antes do lançamento versão final (ROYCE, 1970). No entanto, alguns erros podem não serem encontrados nesta fase, mesmo após uma procura exaustiva. Há erros que aparecem depois de um longo tempo de execução do *software*, que são difíceis de definir suas raízes, estes tipos de erros são muitas vezes chamados de *Heisenbugs* (GRAY; REUTER, 1993).

As falhas relacionadas ao envelhecimento de *sofware* surgem em determinadas condições (por exemplo, falta de recursos do sistema operacional) que não podem ser facilmente reproduzidas (VAIDYANATHAN; TRIVEDI, 2001). O acúmulo dessas falhas pode levar ao travamento do sistema e a falhas totais (HUANG et al., 1995). Portanto, o envelhecimento de *software* consiste na degradação gradual do estado interno do *software* que ocorre devido a erros relacionados com a ativação do envelhecimento (GROTTKE; TRIVEDI, 2007). As principais características de erros relacionadas ao envelhecimento são: (i) provoca acúmulo de erro interno do *software*; (ii) o período de execução do *software* influencia na ativação da falha (GROTTKE; NIKORA; TRIVEDI, 2010).

Devido sua propriedade acumulativa, o envelhecimento de *software* ocorre mais intensamente em processos que executam continuamente durante um longo período. E portanto, os candidatos naturais para exposição ao envelhecimento são os servidores e sistemas embarcados (MATIAS; Freitas Filho, 2006). Esse fenômeno pode ser observado especialmente em infraestruturas de computação nas nuvens, em que o uso de máquinas virtuais e volumes de armazenamento remotos necessitam de operações intensas de memória e disco durante a alocação, reconfiguração ou finalização de máquinas virtuais (ARAUJO et al., 2011; Matos Junior et al., 2012). Dessa forma, o envelhecimento de *software* não está relacionado ao à idade cronológica do *software*, mas à degradação que ele sofre ao longo do tempo de funcionamento.

O envelhecimento de *software* pode prejudicar o sistema de maneiras diferentes. Alguns problemas são: o vazamento de recursos, devido a sistemas com recursos inéditos ou *threads* não finalizadas; fragmentação, corrupção de dados ou erros numéricos (GROTTKE; MATIAS; TRIVEDI, 2008). Uma métrica importante em estudos de envelhecimento de *software* é o *Time To Aging-Related Failure* (TTARF). O TTARF consiste no tempo decorrido entre a inicialização do sistema e a ocorrência de falha relacionada com o envelhecimento de *software* (GROTTKE; MATIAS; TRIVEDI, 2008). A carga de trabalho submetida ao sistema influencia diretamente

no TTARF. Isso acontece porque a quantidade de operações realizadas aumenta o acumúlo de erros e probabilidade de ativação de *bugs* relacionados com o envelhecimento de *software* (COTRONEO et al., 2014). A Figura 2.7 mostra uma representação visual de um sistema com envelhecimento de *software*.

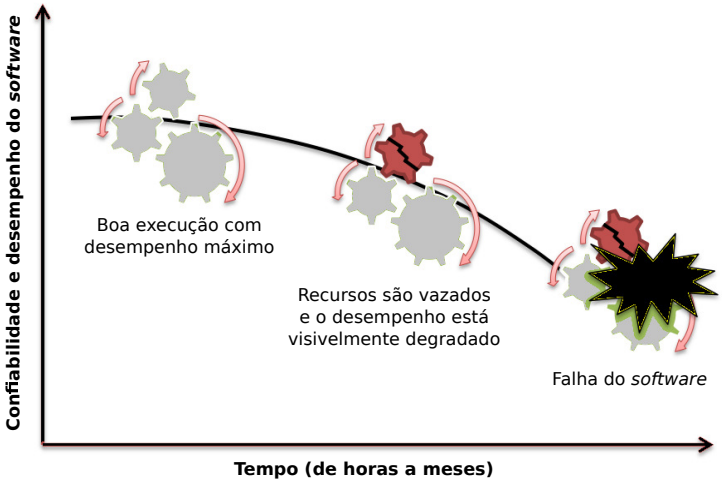


Figura 2.7: Representação de falha de *software* causada pelo envelhecimento

Por outro lado, temos o rejuvenescimento de *software* como uma contramedida para os efeitos do envelhecimento. O rejuvenescimento de *software* consiste em uma reversão proativa do estado do sistema para um estado saudável (HUANG et al., 1995). Algumas ações de rejuvenescimento compreendem operações como coleta de lixo, reindexação de tabelas do kernel e reinicialização de variáveis e estruturas de dados (TRIVEDI; VAIDYANATHAN; GOŠEVA-POPSTOJANOVA, 2000). Normalmente, rejuvenescimento de *software* envolve uma reinicialização do aplicativo ou uma reinicialização completa do sistema. Portanto, o sistema atinge um estado novo, sem acúmulo dos efeitos do envelhecimento *software* (THEIN; PARK, 2009).

Algumas técnicas de rejuvenescimento são descritas abaixo (ALONSO et al., 2013):

- **Reinicialização de aplicação**: É uma simples reinicialização automática da aplicação. Esta técnica de rejuvenescimento para e inicia o processo referente à aplicação. É bastante usada para diminuir os efeitos de envelhecimento em sistemas;

- **Reinicialização de aplicação por *hot standby server***: A aplicação tem uma cópia que está executando simultaneamente em outro servidor. Durante a reinicialização da aplicação as requisições são atendidas pela cópia, o que diminui o *downtime* da técnica anterior. O problema desta técnica é que é necessário adicionar uma réplica da aplicação, adicionar outros componentes como um balanceador de carga o que aumenta o nível de complexidade;

- **Reinicialização do sistema operacional**: Essa técnica reinicializa o sistema operacional por inteiro, o que geralmente envolve reinicialização do *firmware*, re-execução

das rotinas POST (*power-on self test*), reinicialização do *kernel*, reinicialização dos serviços básicos do sistema operacional e finalmente a reinicialização da aplicação;

- **Reincialização rápida do sistema operacional**: A reinicialização padrão do sistema operacional leva a um atraso grande devido a sequência de estágios que precisam ser executados antes que a aplicação possa ser executada. A reinicialização rápida do sistema operacional mantém o estado atual da máquina em relação ao *firmware*, logo não é necessário executar as rotinas POST, sendo necessário apenas reinicializar o *kernel* do sistema operacional, as rotinas básicas e por fim a aplicação;

- **Reinicialização do nó físico**: Essa técnica é baseada em desligar a máquina e ligar novamente, ou seja, literalmente cortar o fornecimento de energia da máquina e reativar o fornecimento novamente.

O rejuvenescimento de *software* é uma ação custosa, pois pode causar a perda de requisições ou até mesmo a indisponibilidade temporária do serviço. É importante definir políticas de rejuvenescimento de *software* para maximizar a disponibilidade do sistema. Uma política amplamente adotada é o agendamento da ação de rejuvenescimento *software* (MELO et al., 2013). Por meio de modelos ou medições, é possível encontrar um agendamento adequado para o acionamento da ação de rejuvenescimento *software*, cujo objetivo principal é aumentar a disponibilidade do sistema (MELO et al., 2013). Diferentemente do tempo de inatividade causado por falhas inesperadas, a inatividade causada pelo rejuvenescimento pode ser agendada para horários em que a carga de trabalho esperada é baixa, a critério do administrador do sistema. Por isso, os custos esperados pela inatividade causada pelo rejuvenescimento são muito menores do que os causados por uma falha inesperada.

Ao longo dos anos, um conjunto significativo de conhecimento foi estabelecido, e a comunidade internacional de pesquisadores na área de envelhecimento e rejuvenescimento de *software* só tem aumentado. Os estudos realizados na área estão relacionados a diversos tipos de aplicações, como o SO Linux (COTRONEO et al., 2010), servidores web como o Apache (MATIAS; Freitas Filho, 2006), Máquina Virtual Java (JVM 5) (COTRONEO; ORLANDO; RUSSO, 2007), computação em nuvem (ARAUJO et al., 2011; MATOS JUNIOR et al., 2012; MELO et al., 2013), dentre outras. Um extenso estudo sobre o envelhecimento e rejuvenescimento de *software* pode ser encontrado em (COTRONEO et al., 2014).

2.6 Considerações finais

Este capítulo apresentou a fundamentação teórica básica para que o leitor possa entender sobre os principais tópicos que compõem esta tese. O *background* sobre modelagem de dependabilidade, bem como a análise de sensibilidade desses modelos, permite entender a aplicação de tais conceitos no planejamento e avaliação de infraestruturas de *mobile cloud computing*.

Modelos e ferramentas de *software* específicos podem desempenhar um papel importante para preencher a lacuna de conhecimento que um administrador de sistema de MCC pode ter em alguns dos conceitos apresentados aqui. Essas ferramentas podem facilitar a aplicação dos métodos propostos, descritos mais adiante, para seu público-alvo.

3

Trabalhos relacionados

Se soubéssemos o que era que estávamos fazendo, não seria chamado pesquisa, seria?

—ALBERT EINSTEIN

O modelo *Mobile Cloud Computing* traz muitos recursos aos dispositivos móveis, os quais são limitados em questão de energia, capacidade de armazenamento, poder de processamento e segurança de dados. A qualidade da experiência de usuário de dispositivos móveis é relativamente aumentada com o uso do modelo MCC, devido ao aumento da capacidade do aparelho.

Os trabalhos encontrados durante a revisão da literatura são descritos neste capítulo. As pesquisas são divididas em duas categorias: **Avaliação de dependabilidade de infraestruturas de MCC** e **Consumo energético em dispositivos móveis**. As seções a seguir não se destinam a fornecer uma visão exaustiva dos trabalhos publicados sobre esses temas, mas sim a apontar avanços significativos que vão para uma direção similar à desta pesquisa, ou proporciona uma base para futuras extensões.

3.1 Avaliação de dependabilidade de infraestruturas de MCC

Os atributos de dependabilidade são essenciais em uma infraestrutura de *mobile cloud computing*, sendo que vários estudos vem sendo desenvolvidos nessa temática. Existem diversas pesquisas na área de MCC, mas somente um pequeno número modela o comportamento da infraestrutura com foco em aspectos de dependabilidade.

MATOS et al. (2015) realizaram uma análise e modelagem de disponibilidade de uma infraestrutura de MCC. O estudo apresenta uma análise de sensibilidade baseada em um modelo analítico hierárquico para analisar os diversos fatores que influenciam a disponibilidade de sistemas MCC. Eles concluíram que a disponibilidade desses sistemas podem ser aumentada consideravelmente reduzindo a indisponibidade de alguns fatores chaves. Este artigo é um dos que mais se assemelha à minha tese, pois também utiliza modelagem hierárquica com diagramas de bloco de confiabilidade e cadeias de Markov para modelar uma infraestrutura de *mobile cloud*

computing e com foco na disponibilidade. A diferença é que o artigo adota três técnicas de análise de sensibilidade para identificar o parâmetro que possui maior impacto na disponibilidade da MCC.

A grande contribuição do artigo é que ele mostra que uma análise de sensibilidade através de derivadas parciais pode não capturar o nível real de impacto para alguns parâmetros em um domínio discreto, como o número de servidores ativos. Sendo que a análise através de outras técnicas como diferença percentual, ou *Design of Experiments* (DoE), atende a essa lacuna.

Já o trabalho de OLIVEIRA et al. (2013) analisa a disponibilidade e o consumo de energia de um ambiente de *mobile cloud computing* através de modelos hierárquicos. Diferentes cenários foram analisados considerando as redes de comunicação WiFi e 3G. Os resultados apresentados demonstraram que a rede 3G consome mais recursos da bateria do que a WiFi, apresentando assim os piores resultados de disponibilidade. O artigo apresenta ainda o tempo média de descarga da bateria para os diferentes cenários. Esse artigo se assemelha à minha tese principalmente por usar modelagem hierárquica (RBD e redes de Petri) e por considerar diferentes fatores de comunicação no tempo de disponibilidade e descarga da bateria.

Um ponto positivo é que os modelos também podem ser adapatados para outros cenários ou de maneira isolada, como é o caso dos modelos de disponibilidade da rede e o de descarga da bateria. Um ponto em desvantagem, é que a ausência de fórmulas fechadas faz com que alterações mais complexas nos modelos necessitem de algum conhecimento sobre redes de Petri.

Já ARAUJO et al. (2016) propuseram modelos analíticos para representar a disponibilidade de sistemas *mHealth* implantados em uma infraestrutura de *mobile cloud computing*. Os autores também utilizaram diagramas de bloco de confiabilidade e cadeias de Markov de tempo contínuo para modelar a infraestrutura. A análise de sensibilidade identificou quais componentes necessitavam de melhorias na disponibilidade. A adoção de estratégias de redundância em diferentes componentes também foi avaliada. Este trabalho se assemelha com a minha tese, pois adota uma arquitetura similar e utiliza as mesmas técnicas de modelagem, mas se concentra em um cenário específico de *mHealth* com componentes específicos, como dispositivos *wearable*.

O maior diferencial do artigo é que além de avaliar o impacto de redundância da rede, ele também avalia o impacto da infraestrutura de nuvem, propondo a utilização de uma *cloudlet* para os dispositivos móveis que estejam próximos fisicamente da infraestrutura computacional. Além disso, os modelos podem ser replicados e adaptados para cenários com alguma similaridade.

PARK et al. (2011) desenvolveram um estudo o qual visa aumentar a tolerância de falhas nos sistemas de *mobile cloud computing* através de predições e análise usando o modelo cadeia de Markov. Neste estudo, os autores propuseram uma técnica que diminue a volatilidade dos sistemas em dispositivos móveis, com a análise dos padrões de estados passados, de forma que possa prever os estados futuros do dispositivo, melhorando a confiabilidade e o desempenho do sistema. Este trabalho se assemelha à minha tese, pois também utiliza cadeias de Markov para modelar o comportamento da *mobile cloud*, além de tentar prever eventuais falhas do sistema. A diferença é que esta tese adotou séries temporais para prever a utilização de recursos e prevenir

eventuais falhar do sistema relacionadas ao envelhecimento de *software*.

A grande vantagem desse artigo é que ele pode ser replicado e adaptado por qualquer pessoa, até mesmo por aquelas que não possuem nenhuma experiência na área de modelagem com cadeias de Markov. Isso é possível em razão das fórmulas fechadas que estão presentes no artigo. Além disso, a mesma estratégia também pode ser adaptada para outros ambientes que necessitem de monitoramento dos recurso computacionais.

Há também pesquisas na área de MCC que levam em consideração outros atributos de dependabilidade não contemplados nesta tese, como integridade e segurança. ITANI; KAYSSI; CHEHAB (2010) desenvolveram um protocolo que garante a integridade de serviços de armazenamento em *mobile cloud computing* e também reduz o consumo de energia do dispositivo móvel. O protocolo proposto pelos autores faz uso de rotinas de criptografia que protegem os dados do usuário enquanto reduz o consumo de energia ao mesmo tempo em que há uma dinâmica operações com dados. Para validar o protocolo desenvolvido por ITANI; KAYSSI; CHEHAB (2010), foi usado modelos analíticos e experimentos de forma que comprovasse a sua eficiência. LOMOTEY; DETERS (2014) desenvolveram um modelo para garantir o acesso de dados médicos em aplicações *mHealth*, de forma que a privacidade do usuário seja preservada. Para alcançar esse objetivo, LOMOTEY; DETERS (2014) sugeriram o uso de um *middleware* orientado a MCC, de forma que sirva como *cache* do sistema, diminuindo assim o tempo de resposta e consequentemente o consumo de energia do dispositivo móvel. No artigo (SALIH; OTHMANE; LESZEK, 2011) foram desenvolvidas estratégias para preservar a privacidade dos dados. Um pacote ativo encapsula os dados sensíveis e uma máquina virtual, dispensando esforço com chaves de descriptografia. Já SUDHA; GANESAN (2013) adotou um algoritmo chamado Criptografia de Curva Elíptica para proteger o acesso aos dados por pessoas não autorizadas em um sistema de saúde. No entanto, tal proteção também produz um impacto sobre o consumo de energia disponível nos *smartphones*.

3.2 Consumo energético em dispositivos móveis

Em WANG et al. (2016), foi proposto um modelo para aumentar a eficiência de uma *mobile cloud computing*, onde foi usado cadeia de Markov e *Grey Theory* para a modelagem dos dispositivos usados em aplicações *mHealth*. O objetivo do estudo foi rastrear o estado de cada componente utilizado pelo sistema *mHealth*, de forma que fosse ativado somente quando necessário, diminuindo assim o consumo de energia e prolongando a vida útil desses componentes. Semelhante à minha tese, esse trabalho também utiliza cadeias de Markov para modelar parte do comportamento de uma *mobile cloud computing*, e um dos objetivos está relacionado ao prolongamento do tempo de vida da bateria dos sensores móveis. Mesmo não tratando especificamente de sensores em minha tese, o uso de baterias é essencial em qualquer dispositivo móvel conectado à MCC.

A grande vantagem desse trabalho é que ele apresenta resultados em tempo real de um

sistema *mHealth*. Por se tratar de um sistema de recomendação, ele considera diversos fatores específicos de sistemas de *mobile cloud computing* e de sistemas de saúde. Além disso, possui um bom potencial de crescimento, pois também considera diferentes situações relacionadas aos cuidados de saúde, aumentando a complexidade da análise. Mesmo considerando especificamente um serviço *mHealth*, a abordagem proposta pode ser adaptada para diferentes áreas que utilizam redes de sensores.

Já em PANNEERSELVAM et al. (2016), os autores elaboraram um *framework* que visa reduzir o tempo em que o dispositivo móvel fica em estado ocioso, encorajando a participação em MCC colaborativas. Em outras palavras, o *framework* desenvolvido sugere a criação de *clusters* com dispositíveis móveis de forma que haja um balanceamento entre consumo de tempo e energia consumida. Os resultados demonstram que os *clusters* de balanceamento de processos são mais eficientes para a execução de tarefas considerando termos energéticos, utilizando de forma eficaz os recursos disponíveis. Os testes foram realizados usando redes 4G/5G. Esse trabalho se assemelha à minha tese pois também adota mecanismos que visam melhorar a utilização da bateria, embora que eu não aborde questões de gerenciamento de processos no dispositivo móvel.

Um ponto positivo é que a estratégia pode ser aplicada a qualquer tipo de serviço que esteja sendo executado, pois adota uma execução colaborativa a nível de processo, independentemente de tipo de aplicação. No entanto, o lado negativo dessa estratégia é que ao criar um *cluster* com dispositivos móveis, inevitavelmente vai aumentar o fluxo de dados transmitidos pela rede. Com isso, em situações que não for possível conectar em redes com alta taxa de transferência, a rede poderá comprometer o desempenho de toda a infraestrutura. Além disso, em situações em que não houver muitos nós do *cluster* ativos e conectados à rede, não será possível adotar a estratégia proposta.

Os autores LIN et al. (2015) avaliaram a performance do acesso a *displays* remotos em *mobile cloud computing*, ou seja, o dispositivo móvel recebe a imagem processada pela MCC. Nesse trabalho os autores focaram no consumo de energia e largura de banda de Internet, mas também levaram em consideração alguns aspectos de usabilidade e experiência de usuário. Os autores adotaram uma abordagem experimental baseada em sessões de usuários reais, empregando diferentes protocolos de acesso remoto e tipos de aplicativos, incluindo jogos. Este trabalho se assemelha à minha tese pois também considera diferentes protocolos de comunicação, mas se limita a apresentar resultados experimentais e sem nenhum modelo que represente o comportamento do ambiente.

Um ponto positivo do trabalho de LIN et al. (2015) é que os resultados com o consumo de energia foram satisfatórios. Contudo, um ponto negativo é que esse modelo não suporta uma alta dinâmica nos conteúdos da tela, devido à alta largura de banda que os protocolos testados usam. Nesse caso, o gargalo criado pela rede pode inviabilizar a utilização dessa estratégia no mundo real.

BOURDENA et al. (2015) desenvolveram um estudo que tinha como principal objetivo

construir um *framework* que ajude a entender o comportamento do consumo de energia de cada dispositivo móvel, através do número de processos que usam a *mobile cloud*. Esse trabalho foi desenvolvido para analisar o consumo de energia dos dispositivos móveis de aplicações de domínio social, como por exemplo redes sociais, onde há um grande fluxo de comunicação com a Internet. BOURDENA et al. (2015) propuseram e avaliaram os experimentos através de simulações e emulações do esquema de *offloading*[1] de aplicações de domínio social. Este artigo se assemelha à minha tese principalmente por propor modelos que representam o comportamento do sistema, sendo que um dos objetivos é avaliar a utilização da bateria de dispostivos móveis.

Um ponto forte desse trabalho é que os autores consideraram aplicações de domínio social, o que abrange uma grande fatia dos usuários de dispositivos móveis atuais. Fatores como a comunicação e o contexto social são utilizados pelos nós móveis com outros parâmetros relacionados com a comunicação, visando a execução eficiente do processo de *offloading* e para manter níveis adequados de qualidade de serviço. Além disso, a comparação com outras estratégias existentes demonstraram que o método proposto é mais eficiente.

CHUNG (2011) avalia o consumo de energia de aplicativos de mensagens instantâneas em duas classes diferentes de serviços e estados, considerando o tempo de inatividade. O autor propõe um modelo de cadeias de Markov para classes de serviço de avaliação da energia. Os resultados demonstraram que o esquema proposto pode superar o esquema convencional para a maioria dos tempos de inatividade. Este trabalho se assemelha à minha tese principalmente pela similaridade do formalismo adotado e também por considerar aplicativos de troca de mensagens.

O ponto forte do trabalho é que os resultados podem ser replicados e adaptados por qualquer pessoa. Além disso, como a avaliação considera características presentes em aplicações com troca de mensagens, os modelos apresentados podem ser adaptados para boa parte das aplicações móveis.

Há também outras pesquisas que utilizam a técnica de *offloading* para ganhos em desempenhos e/ou em consumo de energia. AMORETTI; GRAZIOLI; ZANICHELLI (2015) afirmam que o conceito de probabilidade de *offloading* é um dos pilares do modelo MCC, e as decisões referentes a esse conceito dependem da eficiência do consumo de energia e da performance. O trabalho realizado por AMORETTI; GRAZIOLI; ZANICHELLI (2015) considera dois cenários principais, o consumo de energia de um processo executado totalmente no dispositivo e o consumo de energia de um processo enviado para ser executado na nuvem. Nesse trabalho os autores desenvolveram um *framework* de MCC baseado em *offloading* para programar quando é o melhor momento para fazer a transferência do dispositivo para a nuvem. Para validar o modelo eles compararam os resultados obtidos com os resultados dos experimentos realizados em uma nuvem privada. Já REITER; ZEFFERER (2016) propuseram uma arquitetura de MCC, a qual visa prover uma solução de melhorar o *offloading*. Nesse estudo os autores avaliaram o consumo de energia para validar a solução proposta, onde conseguiram economizar energia mesmo com configurações de alta performance, e na configuração baixa performance, a arquitetura proposta

[1] É o uso de tecnologias complementares para ampliar o poder computacional dos dispositivos móveis.

reduziu o consumo de energia pela metade. Já BARBERA et al. (2013) realizaram um estudo em que consideram os custos de energia e largura de banda de rede no processo de *offloading* em sistemas de *mobile cloud computing*. Esse trabalho estuda a viabilidade de *offloading* em sistemas de MCC e *backups* de *software* e dados em dispositivos móveis em cenários de uso real. Já SHIRAZ et al. (2015) desenvolveram um *framework* para reduzir o consumo de energia do dispositivo móvel durante o *offloading*. O *framework* proposto foca em minimizar as instância de processos durante o *offloading*, de forma que a quantidade de dados transmitidos é reduzida e o consumo de energia também, devido a redução da computação exigida.

Podemos citar ainda outras pesquisas que também focam na redução do consumo de energia em ambientes de *mobile cloud computing*, mas que usam técnicas ou estratégias diferentes das adotadas nesta tese. VALLINA-RODRIGUEZ; CROWCROFT (2012) propõe uma gestão eficiente de energia. Uma grande variedade de interfaces sem fio, e o impacto de sensores e aplicativos de alto desempenho no consumo de energia. Este estudo investiga técnicas de otimização para prolongar a vida da bateria e para despertar o interesse do modelo avaliado aplicado ao consumo de energia. BALASUBRAMANIAN; BALASUBRAMANIAN; VENKATARAMANI (2009) estudou o consumo de energia em dispositivos móveis utilizando as tecnologias GSM, 3G, e Wi-Fi. A pesquisa centrou-se no estudo do estado de cauda, que é a energia que é desperdiçada quando o dispositivo passa para um estado de alto consumo no final de uma típica transferência de dados. Já FRIEDMAN; KOGAN; KRIVOLAPOV (2013) sugeriu um conjunto de padrões e procedimentos operacionais a serem adotadas por desenvolvedores de *software* móvel para preservar recursos energéticos do *smartphone*. E FERRO; POTORTI (2005) apresentou uma comparação entre as tecnologias Wi-Fi e Bluetooth em termos de consumo de energia, analisando os detalhes de implementação de ambas as tecnologias.

3.3 Comparação com os principais trabalhos relacionados

A Tabela 3.1 resume os principais trabalhos relacionados a esta tese que foram mencionados neste capítulo, estabelecendo uma comparação entre eles e a tese com relação a quatro temas: *mobile cloud computing*, atributos de dependabilidade, modelos analíticos/simulação, e consumo de energia da bateria.

Todos os trabalhos apresentados estão inseridos de alguma maneira no contexto de *mobile cloud computing*. Os trabalhos de MATOS et al. (2015); OLIVEIRA et al. (2013); ARAUJO et al. (2016) e PARK et al. (2011) foram apresentados na seção de avaliação de dependabilidade de infraestruturas de MCC por apresentarem características ou com foco principal em atributos de dependabilidade, como disponibilidade e confiabilidade. Alguns deles também apresentaram resultados com foco secundário na redução de energia de bateria. No entanto, todos eles usaram alguma técnica para modelar o sistema, sendo que algumas foram as mesmas adotadas nesta tese.

Já os artigos WANG et al. (2016); PANNEERSELVAM et al. (2016); LIN et al. (2015); BOURDENA et al. (2015) e CHUNG (2011) apresentaram técnicas e/ou estratégias voltadas

Tabela 3.1: Tabela comparativa dos trabalhos relacionados

Autores	MCC	Atributos de dependabilidade	Modelos analíticos /simulação	Consumo de energia da bateria
MATOS et al. (2015)	Sim	Disponibilidade	RBD, CTMC	Sim
OLIVEIRA et al. (2013)	Sim	Disponibilidade	RBD, redes de Petri	Sim
ARAUJO et al. (2016)	Sim	Disponibilidade	RBD, CTMC	Não
PARK et al. (2011)	Sim	Confiabilidade	CTMC	Não
WANG et al. (2016)	Sim	Não	CTMC, *Grey Theory*	Sim
PANNEERSELVAM et al. (2016)	Sim	Disponibilidade	Não	Sim
LIN et al. (2015)	Sim	Não	Não	Sim
BOURDENA et al. (2015)	Sim	Não	FBM	Sim
CHUNG (2011)	Sim	Disponibilidade	CTMC	Sim
Esta tese	Sim	Disponibilidade, Confiabilidade	RBD, CTMC	Sim

especificamente para a redução de energia da bateria. Em alguns casos também consideraram questões relacionadas à disponibilidade, e em outros usaram algum tipo de modelagem para obter os resultados.

3.4 Considerações finais

Este capítulo destacou as principais obras que foram encontradas durante a revisão da literatura sobre os temas mencionados. Embora, é importante enfatizar que esta não é uma visão exaustiva dos artigos publicados e trabalhos de pesquisa relacionados. Pode haver outros artigos e teses que fizeram avanços significativos neste campo, mas com o melhor de nosso conhecimento a combinação das características descritas na Tabela 3.1 é um dos principais fatores que distinguem este trabalho a partir do estado atual da arte.

4
Metodologia

O pessimista vê dificuldade em cada oportunidade; o otimista vê oportunidade em cada dificuldade.

—WINSTON CHURCHILL

Este capítulo apresenta a metodologia de apoio utilizada nesta tese. O mesmo está dividido nas seguintes seções: Visão geral, Pré-avaliação, Avaliação, Melhorias, e Considerações finais.

4.1 Visão geral

Esta seção apresenta uma visão geral da metodologia de apoio utilizada nesta tese, que consiste em três macro-atividades: **Pré-avaliação**, **Avaliação** e **Melhorias**. A **pré-avaliação** está dividida em cinco atividades: entendimento do sistema, definir cenários de interesse, propor modelos de alto nível do sistema, propor submodelos para componentes específicos e obter valores de entrada para os submodelos. Já a **avaliação** está dividida em: avaliar os modelos e executar análise de sensibilidade. Por fim, as **melhorias** consistema em propor melhorias para o sistema.

O fluxograma da Figura 4.1 representa de maneira visual a metodologia de apoio adotada no planejamento de infraestruturas de *mobile cloud computing*. Os retângulos representam cada etapa da metodologia de apoio que foi seguida obedecendo a ordem de execução apontada pelas setas. Somente quando uma etapa é finalizada é que o avaliador segue para a próxima. Já o losango representa uma etapa que pode seguir por dois caminhos diferentes, dependendo do resultado obtido. Nesse caso, a análise é encerrada se os resultados forem satisfatórios e segue para a próxima etapa caso não sejam. Os círculos tracejados representam as possíveis instanciações (estratégias, ações, análises, etc.) de cada etapa da metodologia. Vale destacar que as possíveis instanciações que são apresentadas no fluxograma significam que cada uma delas foi efetivamente adotada em algum momento do estudo, mesmo podendo haver outras

possíveis instanciações que poderiam ser adotadas para cumprir cada etapa. Já as setas horizontais tracejadas apontam para as possíveis instanciações adotadas em cada etapa.

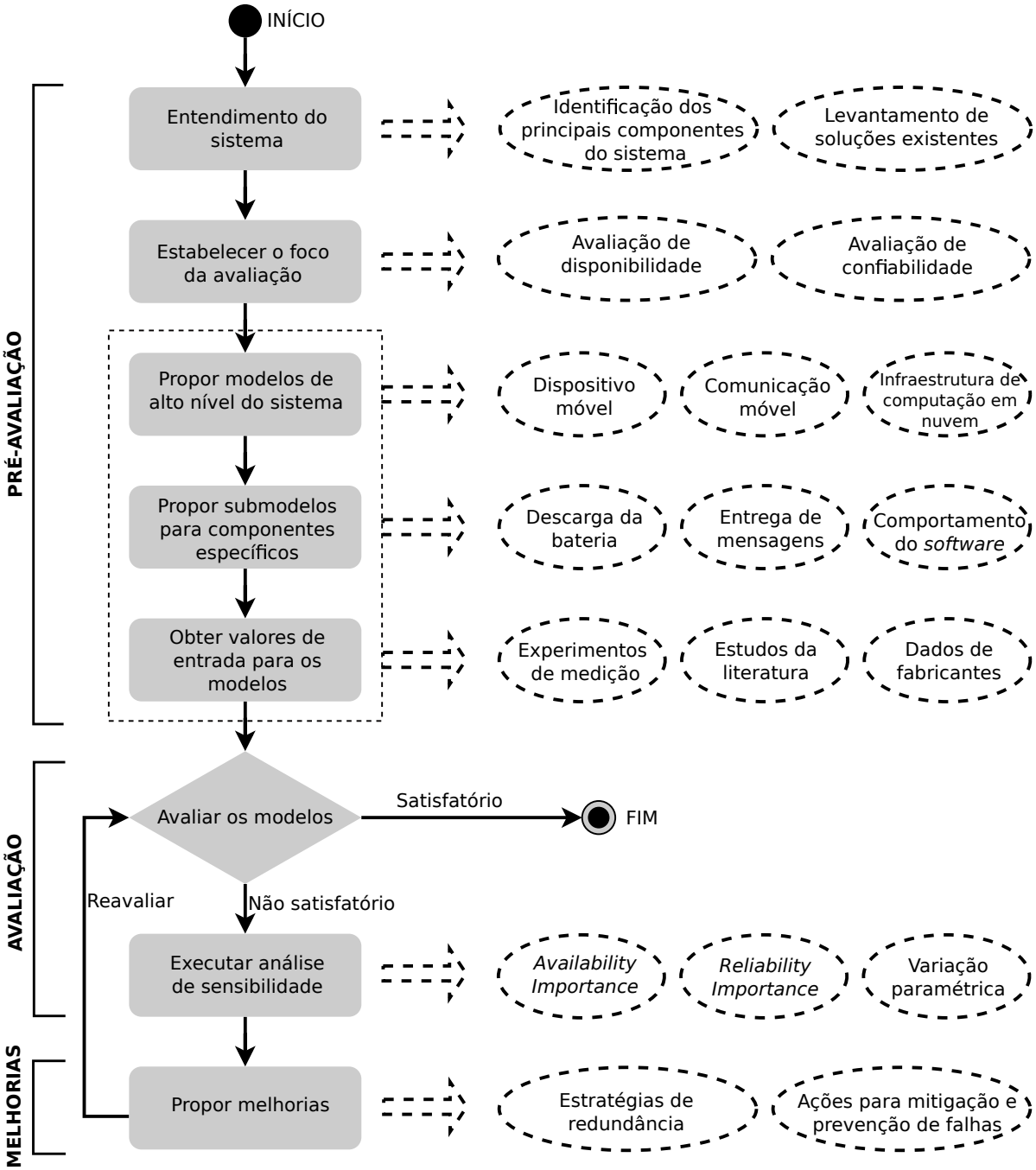


Figura 4.1: Metodologia de apoio para planejamento de infraestruturas de MCC

Cada atividade da metodologia de apoio é descrita em detalhes a seguir.

4.2 Pré-avaliação

Esta seção apresenta as cinco atividades que constituem a pré-avaliação: entendimento do sistema, definir cenários de interesse, propor modelos de alto nível do sistema, propor

submodelos para componentes específicos e obter valores de entrada para os submodelos. O objetivo aqui é apresentar os insumos necessários para a implementação da etapa posterior de avaliação.

Entendimento do sistema:

Para planejar infraestruturas de *mobile cloud computing*, é necessário **entender como o sistema funciona**, **identificando os seus principais componentes** e efetuando um **levantamento das principais soluções existentes**, aplicações e funcionalidades, objetivando uma delimitação da infraestrutura a ser planejada. A compreensão do sistema requer grande atenção e cuidados especiais por parte do avaliador, para assim, evitar erros de interpretação e comprometimento das demais etapas da metodologia. Essa etapa é essencial, pois possibilita tomar conhecimento das técnicas que poderão ser adotadas, adaptadas ou que terão que ser criadas.

- Pré-condições: conhecimentos básicos na área de computação;
- Entradas: leitura de material técnico sobre *mobile cloud computing* e modelagem de sistemas (livros, artigos científicos, sites, fóruns, etc.);
- Ações: identificação dos principais componentes do sistema;
- Produtos: referencial teórico e trabalhos relacionados;
- Pós-condições: entendimento do fucionamento de uma infraestrutura de *mobile cloud computing*; conhecimento sobre técnicas de modelagem de sistemas.

Estabelecer o foco da avaliação:

Logo após, é necessário **estabelecer o foco principal da avaliação** do sistema que está sendo planejado, pois isso irá afetar diretamente nos tipos de modelos que serão criados mais adiante. Nessa etapa, o avaliador deve definir quais cenários e métricas são de interesse, focando principalmente naqueles que possuem maior influência nos níveis de qualidade do serviço oferecido. Por exemplo, objetivando melhorar os índices de dependabilidade do sistema como um todo, cenários que visam avaliar métricas como disponibilidade e confiabilidade podem ser adotados, pois mudanças estruturais podem afetar diretamente tais cenários. Obviamente, métricas como a conectividade da rede e o tempo de descarga da bateria de um dispositivo móvel podem afetar os cenários de **disponibilidade** e **confiabilidade** do sistema. Além disso, a aquisição de equipamentos para melhorias na dependabilidade implica em **custos** adicionais que devem ser administrados adequadamente.

- Pré-condições: entendimento do fucionamento de uma infraestrutura de *mobile cloud computing*;

- Entradas: métricas ou recursos relacionados ao funcionamento de uma infraestrutura de *mobile cloud computing* que podem ser afetados através de mudanças estruturais;
- Ações: estabelecer o foco da avaliação definindo os cenários e métricas que efetivamente serão avaliadas;
- Produtos: relação dos cenários e métricas a serem avaliados;
- Pós-condições: foco da avaliação estabelecido.

Propor modelos de alto nível:

Agora que possíveis lacunas sobre o funcionamento do sistema foram preenchidas e que o foco da avaliação foi estabelecido, podemos **propor modelos de alto nível** para obtermos resultados do sistema como um todo. Dada uma infraestrutura de MCC, é preciso obter uma visão geral do sistema para permitir a criação de um modelo de alto nível. Este é o modelo principal que pode descrever a infraestrutura da *mobile cloud* e a relação global entre os principais componentes da infraestrutura (por exemplo, **dispositivo móvel, comunicação móvel e infraestrutura de computação em nuvem**). RBD é um dos formalismos mais adequados para tal modelo principal, devido à sua concisão para representar grandes sistemas. A modelagem *top-down* induz a criação de modelos condensados, que só incluem o comportamento detalhado para os módulos e subsistemas que são muito bem conhecidos ou que se espera que sejam relevantes para o funcionamento dos sistema.

- Pré-condições: entendimento do fucionamento de uma infraestrutura de *mobile cloud computing*; conhecimento sobre técnicas de modelagem de sistemas; foco da avaliação estabelecido ;
- Entradas: tipo de análise pretendida (por exemplo, disponibilidade, confiabilidade, etc.); lista dos principais componentes ou subsistemas; descrição de dependência ou a interligação entre eles;
- Ações: escolha do formalismo de modelagem; criação do modelo de alto nível;
- Produtos: modelos de alto nível;
- Pós-condições: modelos de alto nível parametrizados e prontos para serem refinados em submodelos, se necessário.

Propor submodelos para componentes específicos do sistema:

Por diversas vezes o modelo geral não consegue expressar características específicas do sistema, então há a necessidade de **propor submodelos** para representar o comportamento interno de **componentes específicos**. Tais submodelos podem imprimir maior granularidade

das características do sistema. Um exemplo disso seria o modelo de **descarga da bateria**, que considera o protocolo de comunicação utilizado, bem como a probabilidade de conectividade presente em diferentes cenários; ou o modelo de **entrega de mensagens**, que sofre influência da rede e da disponibilidade dos componentes do sistema, influenciando no tempo de entrega das mensagens; ou ainda o modelo que representa o **comportamento do *software*** durante a execução de uma ação proativa de prevenção à falhas, o qual permite representar o impacto positivo alcançado pela estratégia de mitigação adotada.

Um submodelo pode compreender outros modelos de nível mais baixo. Caso necessário, diversos outros níveis de modelos podem ser criados para facilitar a análise ou reduzir a complexidade do modelo, por exemplo. No entanto, o avaliador deve ser cauteloso sobre a perda de precisão devido a níveis excessivos nos modelos. Formalismos distintos podem ser escolhidos para cada submodelo, dependendo apenas da adequação para descrever esse subsistema específico e o conhecimento do modelador.

- Pré-condições: modelos de alto nível parametrizados e prontos para serem refinados em submodelos, se necessário;
- Entradas: lista de subsistemas a serem refinados; descrição dos componentes, parâmetros, e funcionamento interno de cada subsistema;
- Ações: escolha dos formalismos de modelagem; criação dos submodelos e definição de como eles serão conectados ao modelo principal;
- Produtos: submodelos e conexões entre o modelo principal e os submodelos;
- Pós-condições: submodelos parametrizados e conectados ao modelo principal.

Obter valores de entrada para os modelos propostos:

Para que os modelos representem adequadamente o comportamento do sistema, alguns dos **valores de entrada** devem ser obtidos através de **experimentos de medição**. Um experimento é um teste de uma série de testes que são executados com o intuito de obter o máximo de informações com o número mínimo de experimentos (JAIN, 1991). Os experimentos são utilizados também para estudar o comportamento de processos e sistemas (MONTGOMERY, 2006). A execução de tais experimentos exige a implantação de um ambiente de testes, e pode estar atrelada também à necessidade de criação de ferramental de apoio. Tal necessidade vai depender da existência de ferramental que suporte a combinação das métricas analisadas, do cenário observado, e do ambiente investigado. Contudo, para aqueles cenários em que a execução de um experimento seja inviável, tais valores deverão ser obtidos através de **estudos da literatura** que disponibilizem valores de componentes com características semelhantes aos da presente tese. Além disso, **dados de fabricantes** também podem ser utilizados para alimentar os modelos propostos. Contudo, é necessário muita cautela, pois os valores fornecidos pelos fabricantes

podem ser superestimados. Nesse caso, o avaliador pode adotar um fator de redução, com base na literatura ou até mesmo de conhecimento empírico, para reduzir os valores e torná-los mais próximos da realidade. Em alguns casos, os valores de entrada de alguns modelos podem ser obtidos através de **outros modelos**, podendo ser submodelos da própria tese ou modelos da literatura.

- Pré-condições: submodelos parametrizados e conectados ao modelo principal; modelos de alto nível parametrizados e prontos;
- Entradas: parâmetros dos modelos propostos;
- Ações: obter os valores de entrada para os parâmetros de cada componente do sistema, seja através da execução de experimentos de medição, de estudos da literatura, de dados fornecidos pelos fabricantes dos componentes ou ainda a partir de outros modelos;
- Produtos: valores de entrada;
- Pós-condições: valores de entrada obtidos; modelos prontos para serem avaliados.

4.3 Avaliação

Já está seção está dividida em duas atividades que constituem a etapa de avaliação: avaliar os modelos e executar análise de sensibilidade. O objetivo aqui é utilizar todo o conhecimento adquirido e material desenvolvido nas etapas anteriores para efetivamente avaliar o sistema.

Avaliação dos modelos:

A **avaliação dos modelos** é a atividade que compara os valores de saída com um valor de referência predefinido através de SLAs ou da expectativa do usuário final ou de administradores de sistema. Quando o valor computado é satisfatório, o processo é finalizado, e é reinicializado somente quando o sistema sofrer alguma modificação. No entanto, se ao menos uma métrica de interesse não alcançou um nível satisfatório, o processo não é finalizado e segue para a próxima etapa.

- Pré-condições: valores de entrada obtidos; modelos prontos para serem avaliados;
- Entradas: valores das métricas de interesse; descrição das expectativas dos usuários finais ou administradores de sistemas, ou SLAs.
- Ações: definir se as métricas estimadas são ou não satisfatórias;

- Produtos: definição (satisfatório ou não satisfatório);
- Pós-condições: resultados satisfatórios; modelos prontos para realizar análise de sensibilidade (quando os valores obtidos não forem satisfatórios, ou para identificar o impacto de mudanças estruturais no sistema).

Análise de sensibilidade:

A etapa de **análise de sensibilidade** é responsável por identificar quais componentes possuem maior impacto nas métricas de interesse do sistema. A fim de encontrar os índices de sensibilidade para um modelo hierárquico, o modelo principal e os submodelos deve, inicialmente, serem avaliados separadamente. Um único método de análise de sensibilidade pode ser utilizado para todos os modelos que compreendem o modelo hierárquico. Métodos distintos também podem ser usados para cada modelo para lidar com restrições de solução específicas ou preferências de análise. Diferentes técnicas como ***availability importance***, ***reliability importance*** e **variação paramétrica** podem ser aplicadas para montar um *ranking* de importância dos componentes. Um único método de análise de sensibilidade pode ser usado para todos os modelos que compõem o modelo hierárquico. Métodos distintos também podem ser usados para cada modelo lidar com restrições de solução específicas ou preferências de análise. A análise efetuada aqui auxilia nas ações que serão tomadas na etapa a seguir.

- Pré-condições: modelos prontos para realizar análise de sensibilidade (quando os valores obtidos não forem satisfatórios, ou para identificar o impacto de mudanças estruturais no sistema);
- Entradas: modelo de alto nível e submodelos; métodos de análise de sensibilidade e ferramentas;
- Ações: executar análise de sensibilidade para o modelo de alto nível e para os submodelos;
- Produtos: índices de sensibilidade para os modelos;
- Pós-condições: identificação dos componentes mais importantes do sistema.

4.4 Melhorias

Por fim, etapa final da metodologia é composta por uma única atividade, que é propor melhorias. O objetivo é adotar os resultados que foram obtidos na etapa anterior de avaliação e usá-los para tomar decisões que melhorem os níves de disponibilidade e confiabilidade do sistema de MCC.

Propor melhorias:

Por fim, diversas **melhorias** podem ser **propostas** para alcançar altos níveis de dependabilidade. **Estratégias de redundância** como TMR, *hot-standby*, *warm-standby*, e *cold-standby* podem ser adotadas para melhorar a disponibilidade devido a falhas de *hardware*. Quanto às falhas de *software*, **ações para mitigação e prevenção de falhas** podem impedir, por exemplo, falhas causadas pelo envelhecimento de *software*, maximizando os níveis de qualidade de serviço.

- Pré-condições: identificação dos componentes mais importantes do sistema.
- Entradas: estratégias de melhorias do sistema para aumentar os níveis de disponibilidade devido a falhas de *hardware* ou de *software*;
- Ações: avaliação do impacto da implantação dos diferentes tipos de redundância de *hardware* e de ações para mitigação e prevenção de falhas de *software*;
- Produtos: melhores níveis de qualidade de serviço;
- Pós-condições: Reavaliar os modelos para identificar o impacto das melhorias na dependabilidade do sistema.

4.5 Considerações finais

Este capítulo apresentou a metodologia utilizada para o planejamento de infraestruturas de *mobile cloud computing* com foco principal na avaliação de requisitos de disponibilidade e confiabilidade. Através de um conjunto de etapas que aborda desde o entendimento do funcionamento básico do sistema até a proposição de melhorias em componentes do sistema, este capítulo proporciona detalhes do planejamento da avaliação que podem ser replicados por outros pesquisadores.

5
Modelos

Lute e lute novamente, até cordeiros virarem leões.
—RIDLEY SCOTT

Esta tese de doutorado propõe modelos para o planejamento de infraestruturas de *mobile cloud computing*, com foco em aspectos de dependabilidade. Esta abordagem foca em modelos hierárquicos e possibilita a identificação de pontos de falhas de maior impacto para o sistema, bem como propõe melhorias na infraestrutura para aumentar os níveis de qualidade do sistema.

A arquitetura geral da infraestrutura de MCC considerada nesta tese é muito semelhante a diversas situações, tais como: um corredor de maratona que tem seus sinais vitais monitorados durante uma competição; um carro em movimento usando aplicativos em nuvem para ajudar na localização do motorista; um sistema *mHealth*, que monitora os sinais vitais do paciente dentro e fora do hospital; entre outros. A Figura 5.1 mostra uma visão geral de uma infraestrutura de *mobile cloud computing*.

Para o propósito geral deste estudo e para facilitar a explicação da interação dos principais componentes do sistema, estamos considerando um serviço *mHealth* implantado em uma infraestrutura de ***mobile cloud computing***, a qual é composta por três componentes principais: **dispositivos móveis**, **comunicação móvel**, e **infrastrutura computacional**. Os **dispositivos móveis** fornecem o serviço móvel para o usuário (paciente ou equipe médica). Cada consulta é requisitada a uma infraestrutura computacional através da comunicação móvel; a **comunicação móvel** é a maneira através da qual todas as informações serão transportadas. Neste estudo, nós consideramos dois tipos de conexões: **rede local**[1] e **rede móvel**; a **infraestrutura computacional** é responsável pelo armazenamento e processamento de dados. Para o propósito desse estudo, iremos considerar que o sistema utiliza uma infraestrutura de computação em nuvem.

Uma *mobile cloud* usa infraestruturas de computação em nuvem para fornecer seus serviços. Alguns *frameworks*, como Eucalyptus, OpenStack, e OpenNebula podem ser usados para implementar uma nuvem no estilo IaaS. Os *frameworks* de computação em nuvem que

[1] Por questões de simplicidade, iremos nos referir a uma *Wireless Local Area Network* (WLAN) como rede local.

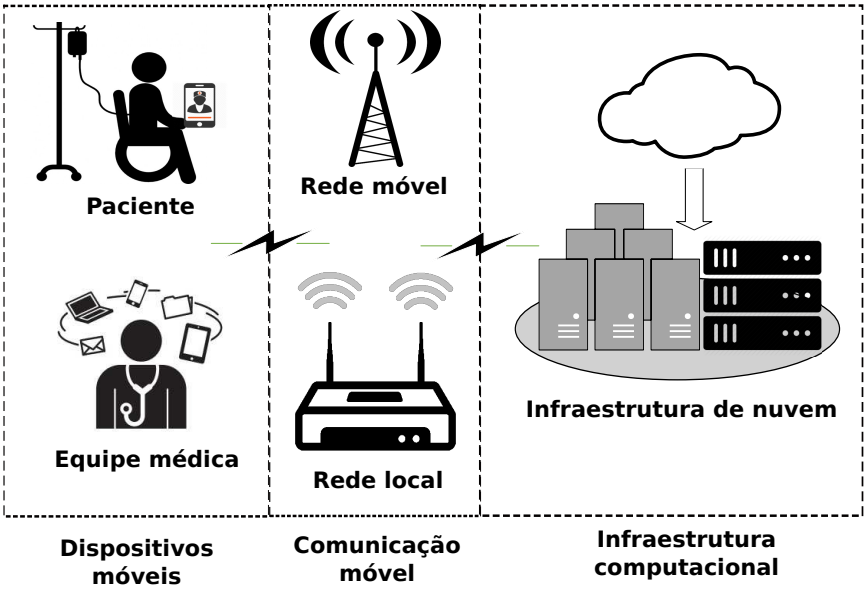


Figura 5.1: Visão geral de uma infraestrutura de *mobile cloud computing*

implementam um estilo privado/híbrido do modelo IaaS têm componentes similares. Nosso trabalho considera as principais características de várias estruturas em nuvem para que os componentes recebam nomes genéricos, enfatizando que o modelo não é restrito a um *framework* específico.

A seguir, iremos apresentar os quatro conjuntos de modelos propostos. O primeiro conjunto tem por objetivo principal avaliar a disponibilidade e a confiabilidade de uma infraestrutura de MCC considerando diferentes arquiteturas e mecanismos de redundância, bem como avaliando o custo de implantação e provisionamento de uma nuvem privada, e comparando com uma nuvem pública. O segundo conjunto considera a existência de falhas na MCC relacionadas ao envelhecimento de *software*, bem como o impacto com a adoção de políticas de rejuvenescimento de *software*. Já o terceiro conjunto avalia principalmente o tempo de vida da bateria com base no tempo médio de descarga considerando diferentes tipos de cenários de conectividade e diferentes protocolos de comunicação. Por fim, o quarto conjunto modela o tempo médio de entrega de mensagens em uma infraestrutura de MCC também considerando difrentes tipos de conectividade.

5.1 Modelos de disponibilidade e confiabilidade *vs* custo de implantação e provisionamento

O propósito deste estudo é avaliar a disponibilidade e confiabilidade de uma infraestrutura de *mobile cloud computing* considerando diferentes arquiteturas e mecanismos de redundância, bem como avaliando o custo de implantação e provisionamento de uma nuvem privada, e comparando com uma nuvem pública. Com isso, considerou-se que um sistema de saúde onde o paciente está acamado e, portanto, sem sair das instalações do hospital, ou seja, o paciente possui uma conexão de Internet estável. Já no lado da equipe médica, não há garantias de que o

sistema está sempre disponível, pois o agente de saúde estará em constante movimento. Portanto, só iremos considerar a disponibilidade/confiabilidade do sistema a partir do dispositivo móvel utilizado pela equipe de saúde que usa uma comunicação móvel para acessar as informações do paciente armazenadas na nuvem. Uma visão geral do sistema é representada através da Figura 5.2.

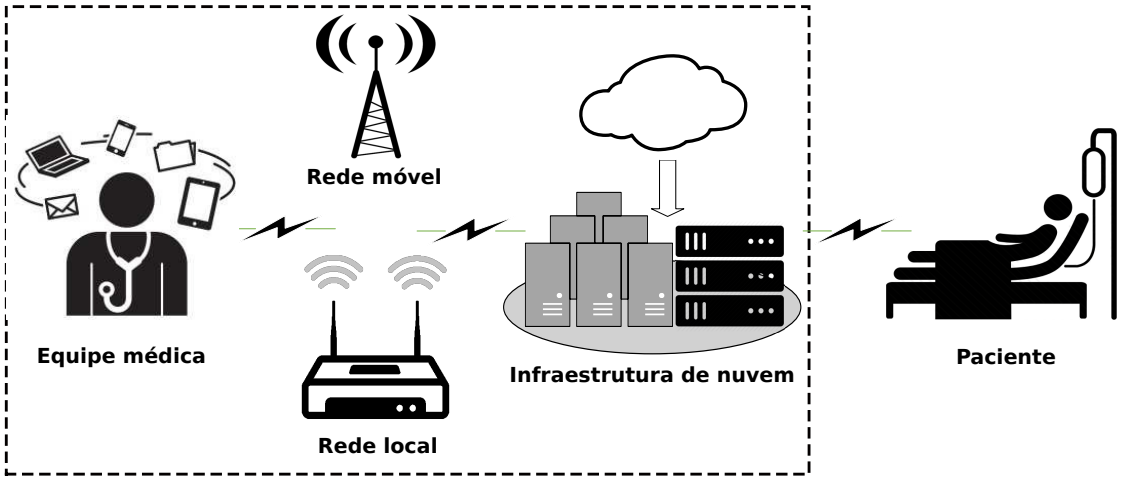


Figura 5.2: Visão geral de um sistema *mHealth*

Considerando uma arquitetura não redundante (*baseline*), a infraestrutura de nuvem é composta por alguns componentes básicos de gestão: um Gerente da Infraestrutura (GI), um Gerente de Armazenamento (GA), e cinco Gerentes dos Nós (GN). O GI é o *front-end* de toda a infraestrutura de nuvem, e é responsável por disponibilizar e gerenciar os recursos virtualizados; o GA fornece armazenamento em bloco para uso das instâncias de máquinas virtuais; e, finalmente, o GN controla o ciclo de vida de instâncias em execução no nó. Ele é usado para instanciar as máquinas virtuais que efetivamente vão expandir o poder de processamento dos dispositivos móveis, ou seja, é responsável pelo poder de processamento da nuvem.

Todo o sistema está disponível se o dispositivo móvel estiver disponível, se ao menos um tipo de comunicação móvel também estiver disponível, e se a infraestrutura de nuvem também estiver disponível, ou seja, todos os três componentes devem estar funcionando corretamente. Para o lado da nuvem, note que pelo menos um dos cinco nós tem de estar disponível para o funcionamento adequado da nuvem. Por outro lado, uma única falha no GI ou GA faz com que a nuvem privada fique indisponível.

Para representar tal ambiente e estimar a dependabilidade do sistema, alguns modelos analíticos foram criados com auxílio da ferramenta Mercury (SILVA et al., 2015), e as fórmulas fechadas foram obtidas com a ferramenta Mathematica[2]. Os resultados foram obtidos por análise transiente.

O sistema geral da infraestrutura de MCC é representado por um modelo RBD, mostrado na Figura 5.3. Cada subsistema possui métricas de disponibilidade calculadas através dos

[2]Disponível em: www.wolfram.com/mathematica

submodelos correspondentes. Consideramos que o MTTF e o MTTR de cada componente são exponencialmente distribuídos. O mesmo pressuposto é usado para todos os submodelos.

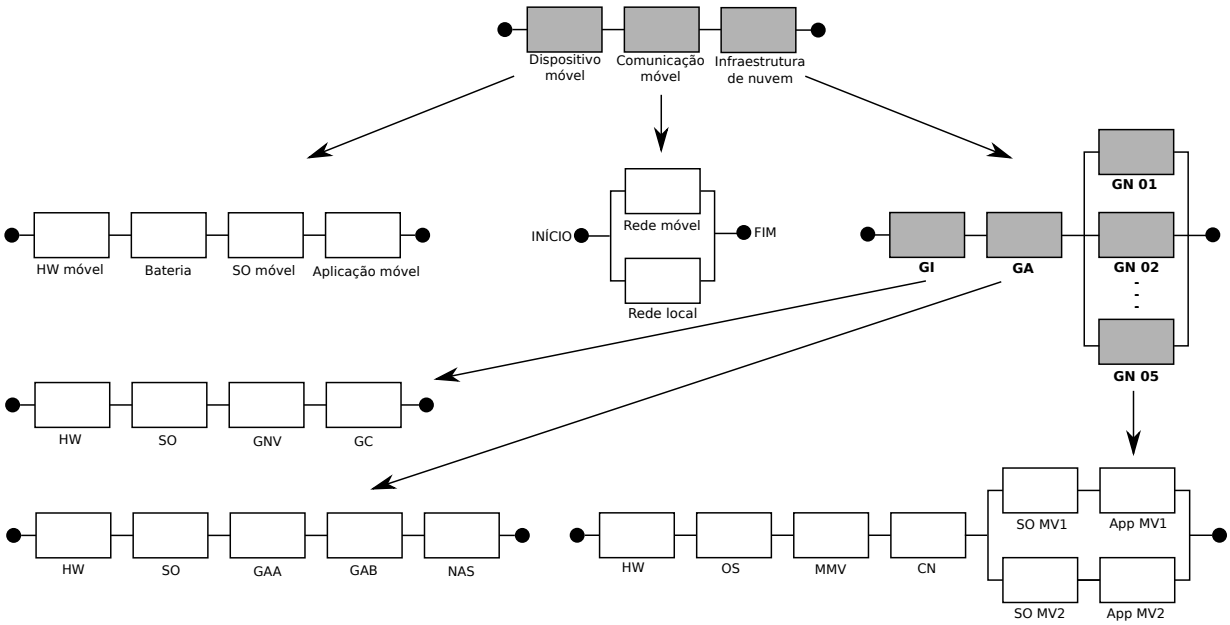


Figura 5.3: Modelo RBD hierárquico para arquitetura básica

A fórmula fechada para a disponibilidade do sistema (A_s) é expressada pela Equação 5.1. Cada componente A_n nesta equação é calculado a partir da avaliação do respectivo submodelo, que também pode ser feito através de fórmulas fechadas, se for possível obtê-las, ou através de solução numérica.

$$A_s = A_{dm} \times A_{cm} \times A_{in} \tag{5.1}$$

O lado do dispositivo móvel é composto pelos componentes ***hardware* móvel**, **bateria**, **SO móvel**, e **aplicação móvel**. O ***hardware* móvel** se refere ao dispositivo físico; **bateria** é a bateria presente nos dispositivos móveis; **SO móvel** é o sistema operacional em execução no dispositivo móvel; a **aplicação móvel** representa a aplicação que se beneficia dos recursos da nuvem para melhorar seu desempenho e prestar um melhor serviço. Eles não são expandidos em submodelos, mas a disponibilidade de todo o dispositivo móvel é calculada a partir da Equação 5.2.

$$A_{dm} = A_{hw\_movel} \times A_{bateria} \times A_{so\_movel} \times A_{app\_movel} \tag{5.2}$$

O lado da **comunicação móvel** é composto somente pelos componentes **rede local** e **rede móvel**. Para o propósito desse estudo, estamos considerando apenas falhas e reparos dos componentes da infraestrutura, sem considerar questões de área de cobertura e/ou bloqueios do sinal. Portanto, a disponibilidade é calculada usando a Equação 5.3. Os parâmetros λ_{rl} e μ_{rl} são os valores de falha e de reparo da rede local, enquanto λ_{rm} and μ_{rm} são os valores de falha e de

reparo da rede móvel. Falhas simultâneas em ambos os componentes deixam a comunicação indisponível, e consequentemente, todo o sistema.

$$A_{cm} = (1 - (1 - A_{rl}) \times (1 - A_{rm})) \tag{5.3}$$

O lado da **infraestrutura de nuvem** é dividido em três submodelos: **GI**, **GA**, e **GN**. A disponibilidade é calculada a partir da Equação 5.4

$$A_{in} = A_{gi} \times A_{ga} \times (1 - (1 - A_{gn})^5) \tag{5.4}$$

O submodelo **GI** possui quatro componentes: *HW*, *SO*, *GNV*, e *GC*. O *Hardware* (HW) representa os componentes físicos que compõem o *hardware*; O Sistema Operacional (SO) é o sistema operacional instalado na máquina; GNV se refere ao *webservice* do gerente da nuvem, ou seja, o principal componente responsável pelo gerenciamento de toda a infraestrutura de nuvem; GC representa o gerente do *cluster*, que é o *front-end* do *cluster*, e gerencia um conjunto de nós. A disponibilidade do submodelo GI é calculada através da Equação 5.5.

$$A_{gi} = A_{hw} \times A_{so} \times A_{gnv} \times A_{gc} \tag{5.5}$$

O submodelo do **GA** é dividido é cinco componentes: *HW*, *SO*, *GAA*, *GAB*, e *NAS*. O HW e o SO representam a máquina física e o sistema operacional, respectivamente. O Gerente de Armazenamento baseado em Arquivo (GAA) representa um serviço de armazenamento de dados baseado em arquivo; O Gerente de Armazenamento baseado em Bloco (GAB) fornece armazenamento para uso das instâncias de máquinas virtuais; NAS (do inglês, *Network-Attached Storage*), é um dispositivo que proporciona o armazenamento de dados através da rede para os gerentes de armazenamento da nuvem. A disponibilidade da submodelo GA é calculada através da Equação 5.6.

$$A_{ga} = A_{hw} \times A_{so} \times A_{gaa} \times A_{gab} \times A_{nas} \tag{5.6}$$

Cada nó no subsistema **Gerente dos Nós (GN)** possui seis componentes: *HW*, *SO*, *MMV*, *CN*, *SO_MV*, e *App_MV*. HW e SO têm o mesmo significado dos modelos anteriores; O Monitor de Máquina Virtual (MMV) representa um *software* que cria e executa as máquinas virtuais, também conhecido como *hypervisor*; o Controlador do Nó (CN) é o *webservice* responsável pelo gerenciamento do nó, é um componente que controla o ciclo de vida das instâncias em execução no nó; *SO_MV*, refere-se ao sistema operacional em execução em cada Máquina Virtual (MV) instanciada no nó; e *App_MV* representa a aplicação que é executada na MV e se comunica com o dispositivo móvel a fim de fornecer o serviço de nuvem móvel. Consideramos que duas VMs estão rodando em cada nó. A disponibilidade da submodelo GN é calculada através da Equação 5.7.

$$A_{gn} = A_{hw} \times A_{so} \times A_{mmv} \times A_{cn} \times (1-(1-(A_{so\_mv1}) \times (A_{app\_mv1})) \times (1-(A_{so\_mv2}) \times (A_{app\_mv2}))) \quad (5.7)$$

Quatro técnicas são aplicáveis para alcançar uma boa dependabilidade dos sistemas: a prevenção da falha, remoção da falha, tolerância a falhas, e previsão de falhas. Ao contrário de outras técnicas, a tolerância a falhas se destina a manter o fornecimento do serviço corretamente na presença de falhas. É geralmente implementada pela detecção de erros e subsequente recuperação do sistema (AVIZIENIS; LAPRIE; RANDELL, 2001). Neste contexto, estratégias de redundância são comumente usadas para melhorar a disponibilidade dos sistemas. No entanto, uma estratégia de redundância pode aumentar os custos e a complexidade do sistema além de um nível aceitável. Por isso, é necessário um estudo detalhado do sistema para aplicar a estratégia mais indicada. Foram considerados quatro tipos de estratégias de redundância: TMR, *hot-standby*, *warm-standby*, e *cold-standby*.

O modelo *Triple Modular Redundancy* (TMR) é composto por três módulos. Ele possui características que permitem a falha de um dos três módulos sem incorrer em uma falha do sistema. Foi adotado o bloco do tipo *k-out-of-n*, onde o sistema ainda está disponível se duas das três máquinas estiverem disponíveis. A Equação 5.8 pode ser usada para calcular a disponibilidade do modelo TMR.

$$A_{tmr}(t) = 1 - P_{n-k+1}(t) \quad (5.8)$$

onde $P_i = \lim_{n\to\infty} P_i(t)$. Maiores detalhes sobre o cálculo da disponibilidade em sistemas do tipo *k-out-of-n* pode ser encontrado no livro publicado por KUO; MING (2002).

A estratégia *hot-standby* (LEE; ANDERSON, 1990) adota dois componentes no sistema. Quando o componente principal falha, um componente de reposição assume imediatamente o serviço, sem demora significativa. O componente de reposição tem a mesma taxa de falha que o componente ativo. É geralmente adotada em sistemas que necessitem de disponibilidade elevada. Um modelo RBD com dois blocos paralelos representa a estratégia de redundância *hot-standby*. A Equação 5.9 pode ser usada para calcular a disponibilidade para do modelo *hot-standby*.

$$A_{hot} = 1 - \prod_{i=1}^{2}(1 - A_i) \quad (5.9)$$

onde A_i é a disponibilidade do estado estacionário do componente principal ou componente de reposição, o que significa que o sistema está operacional, se pelo menos um componente estiver funcionando.

A redundância *warm-standby* (LEE; ANDERSON, 1990) também é composta por dois componentes: um ativo (principal), e um não-ativo (reserva). Quando o componente principal falha, o componente reserva é ativado para minimizar a indisponibilidade do sistema. O processo de ativação leva pouco tempo. Esse período de tempo é chamado de *Mean Time to Activate*

(MTTA) (GUIMARãES; MACIEL; JR., 2013) com taxa α. O componente principal e o reserva possuem taxa de falha λ e taxa de reparo μ. No entanto, o componente reserva possui uma taxa de falha ϕ menor quando não está ativado ($0 \leq \phi \leq \lambda$). Nós também assumimos que a taxa α é 50% menor que λ (GUIMARãES; MACIEL; JR., 2013). As taxas λ e μ são obtidas a partir do MTTR e MTTF computados a partir do respectivos modelos RBD do componente que está sendo estudado.

O modelo *warm-standby* para sistemas redundantes (DANTAS et al., 2012) é apresentado na Figura 5.4. Esse modelo possui cinco estados: UW, UD, DU, DW, e DD. O sistema está disponível nos estados UW, UD e DU (estados brancos) e está indisponível nos estados DW e DD (estados cinza).

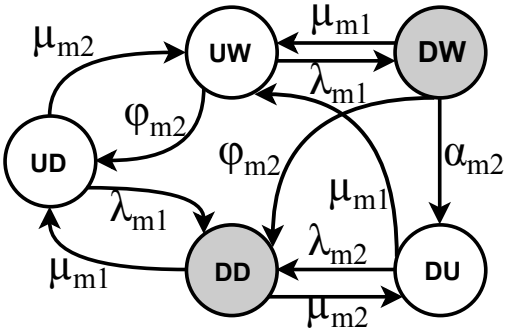


Figura 5.4: Modelo *warm-standby* para sistemas redundantes.

No estado UW, o componente principal (M1) está funcionando (up) e o componente reserva (M2) está em uma condição de espera (*waiting*). Quando M1 falha (λ_{m1}), o sistema vai para o estado DW, onde M2 está aguardando para ser ativado depois da falha do componente principal. Nesse estado, M1 pode ser reparado (μ_{m1}), retornando o sistema para o estado UW, ou M2 pode ser ativado (α_{m2}) e o sistema vai para o estado DU, onde o componente reserva deixa a condição de espera e assume o papel de gerente do sistema.

No estado DU, considerando que M1 ainda está em modo de falha, M1 ainda pode ser reparado e o sistema retorna novamente para o estado UW, ou M2 também pode falhar (λ_{m2}), levando o sistema para o estado DD. Agora, no estado DD, como estamos assumindo que todas as máquinas possuem a mesma configuração, não haverá prioridade no reparo de qualquer máquina. Portanto, qualquer uma pode ser reparada primeiro, ou seja, o sistema pode ir para o estado UD ou DU. Se M1 for reparada (μ_{m1}), o sistema vai para o estado UD. Depois disso, se M2 também for reparada (μ_{m2}), o sistema retorna para o estado UW. A taxa de falha ϕ_{m2} representa uma taxa de falha menor atribuída a M2 quando não estiver ativado.

A Equação 5.10 pode ser usada para calcular a disponibilidade do modelo *warm-standby*.

$$A_{warm} = \frac{2\mu(\lambda\phi + (\mu+\phi)^2 + \alpha(\lambda+\mu+\phi))}{2\mu^2(\lambda+\mu) + (\lambda+2\mu)^2\phi + (\lambda+2\mu)\phi^2 + \alpha(\lambda^2 + 2\mu(\mu+\phi) + \lambda(2\mu+\phi))} \quad (5.10)$$

A estratégia *cold-standby* (LEE; ANDERSON, 1990) também é composta por dois

componentes: um ativo (principal), e um não-ativo (reserva). O componente reserva é ativado quando o componente principal falha. Nesta estratégia, o componente de reposição leva mais tempo para ativar porque geralmente não possui qualquer *software* ou dados instalados. Como o componente de reposição está desligado, considera-se que ele não irá falhar até tornar-se ativo e operacional (SHOOMAN, 2002). Aqui, o MTTA da estratégia *warm-standby* é maior que é a *cold-standby* ($0 \leq \alpha_{cold} < \alpha_{warm}$). As taxas λ e μ são obtidas a partir do respectivo modelo do componente que a estratégia de redundância está sendo aplicada.

O modelo *cold-standby* para sistemas redundantes é apresentado na Figura 5.5. Esse modelo possui cinco estados: UO, UD, DU, DO, e DD. O sistema está disponível nos estados UO, UD e DU (estados brancos) e indisponível nos estados DO e DD (estados cinza).

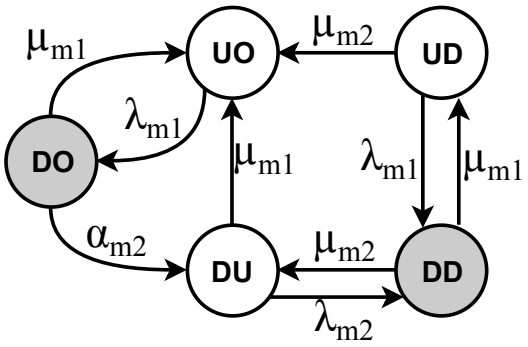


Figura 5.5: Modelo *cold-standby* para sistemas redundantes.

No estado UO, o componente principal (M1) está funcionando (up) e o componente reserva (M2) está desligado (*off*). Quando M1 falha (λ_{m1}), o sistema vai para o estado DO, onde M2 ainda está desligado, aguardando para ser inicializado e configurado para assumir a gestão do sistema após a falha de M1. Nesse estado, M1 pode ser reparado (μ_{m1}), retornando o sistema para o estado UO, ou M2 pode ser inicializado, preparado, e ativado (α_{m2}), e o sistema vai para o estado DU, onde o componente reserva sai da condição de desligado e assume o papel de gestor do sistema. Agora, no estado DU, enquanto M1 ainda está em modo de falha, M1 pode ser reparado e o sistema retorna novamente para o estado UO, ou M2 também pode falhar (λ_{m2}), levando o sistema para o estado DD. No estado DD, como estamos assumindo que todas as máquinas têm a mesma configuração, não vamos priorizar o reparo de qualquer máquina e, portanto, qualquer uma pode ser reparada primeiro, ou seja, o sistema pode ir para os estados UD ou DU. Se M1 for reparado (μ_{m1}), o sistema vai para o estado UD. Depois disso, se M2 também for reparado (μ_{m2}), o sistema retorna para o estado UO.

A Equação 5.11 pode ser usada para calcular a disponibilidade do modelo *cold-standby*.

$$A_{cold} = \frac{2\mu(\mu^2 + \alpha(\lambda + \mu))}{2\mu^2(\lambda + \mu) + \alpha(\lambda^2 + 2\lambda\mu + 2\mu^2)} \tag{5.11}$$

Quantificando os custos de implantação e provisionamento de uma nuvem privada:

Para calcular o custo total de instalação e manutenção de uma nuvem privada, o **Custo Total (CT)** é representado pelos custos de aquisição e custos operacionais dos equipamentos da infraestrutura de **Tecnologia da Informação (TI)** e da **Infraestrutura de Refrigeração (IR)**. Nesta tese, estamos considerando as seguintes despesas principais: custo com a **aquisição de equipamentos** (σ), que representa as despesas com a aquisição dos equipamentos que compõem a infraestrutura de nuvem, e a infraestrutura de refrigeração, que é responsável por extrair o calor produzido pela infraestrutura de TI; os custos referentes ao **consumo de energia** (ψ) gasto durante o funcionamento desses equipamentos e gasto com o resfriamento do ambiente; e custos de **reparo** (δ), que incluem os custos com a reparação, substituição e mão de obra para execução dos serviços de reparo e manutenção.

O custo de aquisição de equipamentos do sistema (σ_s) corresponde aos recursos financeiros necessários para comprar os equipamentos de TI para a infraestrutura de nuvem privada, e da infraestrutura de resfriamento. O custo do equipamento é calculado a partir da Equação 5.12.

$$\sigma_s = (\omega_{TI_i} \times \xi_{TI_i}) + (\omega_{IR_i} \times \xi_{IR_i}), \tag{5.12}$$

onde ω_{TI_i} refere-se ao número de computadores a ser adquirido para toda a infraestrutura de TI, e ξ_{TI_i} é o custo de um único computador; ω_{IR_i} representa o número de equipamentos de refrigeração, e ξ_{IR_i} é o custo de um único equipamento de refrigeração.

Considerando a variedade de cenários que serão estudados nesta tese, a taxa do consumo de energia será calculada separadamente para cada máquina γ_{TI_i}. Para essa avaliação o γ_{TI_i} de cada máquina é calculado a partir da Equação 5.13, adaptada de (CALLOU et al., 2013).

$$\gamma_{TI_i} = \rho_{input_i} \times (\beta_{TI_i} + \alpha \times (1 - \beta_{TI_i})), \tag{5.13}$$

onde ρ_{input_i} é a potência de entrada em kilowatt (kW) para cada máquina i; β_{TI_i} refere-se à quantidade de máquinas ativas; e α é um fator que representa a fração de energia que continua a ser consumida mesmo após a falha do componente. Mesmo com valores semelhantes, em alguns casos β não possui o mesmo valor de ω_{TI_i} (onde $0 < \beta_{TI_i} \leq \omega_{TI_i}$).

Para os cenários sem redundância, o β_{TI_i} é calculado a partir de $\beta_{TI_i} = A_i = \frac{\lambda_i}{\lambda_i + \mu_i}$.

Assim, a taxa instantânea de consumo de energia de toda a infraestrutura de TI é calculada a partir da Equação 5.14.

$$\gamma_{TI} = \sum_{i=1}^{n} \gamma_{TI_i}, \tag{5.14}$$

onde γ_{TI_i} é o consumo de energia de cada máquina. Os valores são obtidos a partir da Equação 5.13.

O consumo energético de todas as máquinas da infraestrutura é calculado a partir da Equação 5.15.

$$\psi_{TI} = T \times \gamma_{TI}, \tag{5.15}$$

onde T é tempo observado em horas; e γ_{TI} é a energia consumida pelo sistema inteiro.

O custo total de energia para a infraestrutura de TI em um período T pode ser calculada a partir da Equação 5.16, ou da Equação 5.17.

$$\psi_{TI_t} = \xi_{power} \times \psi_{TI}, \tag{5.16}$$

$$\psi_{TI_t} = \xi_{power} \times T \times \gamma_{TI}, \tag{5.17}$$

onde ξ_{power} é o custo médio de eletricidade por quilowatt-hora; T é o tempo observado em horas; e ψ_{TI} é o consumo de energia de todo o sistema de TI, e os seus valores são obtidos a partir da Equação 5.15.

Para o custo de energia da infraestrutura de refrigeração, estamos assumindo que ele consome cerca de 40% do consumo total de energia da infraestrutura de TI (CALLOU et al., 2012). Assim, o custo total de energia consumida pela infraestrutura de refrigeração em um tempo T pode ser calculado a partir da Equação 5.18.

$$\psi_{ir_t} = \psi_{TI_t} \times \tau \tag{5.18}$$

onde ψ_{TI_t} é o custo total de energia para infraestrutura de TI em um tempo T, que é obtida a partir da Equação 5.16; e τ refere-se ao percentual de consumo de energia que é necessária para a infraestrutura de refrigeração remover o calor produzido pela infraestrutura de TI, ou seja, nesta tese $\tau = 0.4$ (CALLOU et al., 2012).

Outro custo que também é considerado neste trabalho é o montante gasto no reparo das máquinas. As falhas de *software* não foram adicionadas ao custo de reparo porque estamos considerando que os profissionais de TI são capazes de corrigir os problemas de *software*. Para isso, foram incluídas apenas a falhas de *hardware*, uma vez que pode necessitar de uma peça de reposição. Este custo é avaliado a partir da Equação 5.19.

$$\zeta_r = \omega_{TI_i} \times \omega_f \times \xi_r \tag{5.19}$$

onde ω_{TI_i} refere-se ao número de computadores adquiridos para cada cenário; ω_f representa o número de falhas de *hardware* em um tempo de observação T, obtido a partir de $\omega_f = \frac{\lambda_{hw}}{T}$; ξ_r é o preço médio de cada reparo, já incluindo as aquisição e substituição da peça danificada.

Finalmente, o cálculo do custo total para instalar e manter uma nuvem privada em um período de tempo T, também está considerando os gastos com um profissional de TI (ζ_{mp}) que é responsável pelo gerenciamento da nuvem e capaz de fazer qualquer reparo de *software*. Consideramos também o custo do aluguel do espaço físico onde a nuvem privada será alocada (ζ_{room}). Mesmo que o espaço físico seja próprio, estamos considerando a existência do custo de oportunidade, que, neste caso, refere-se ao custo econômico causado pela renúncia do espaço. Os custos totais podem ser calculados com a Equação 5.20.

$$\Omega = \psi_{TI_t} + \psi_{ri_t} + \zeta_r + \zeta_{mo} + \zeta_{room} \tag{5.20}$$

onde ψ_{IT_T} é o custo total de energia para infraestrutura de TI em um tempo T; ψ_{ri_t} representa o custo de energia da infraestrutura de refrigeração em um tempo T; ζ_r é o montante gasto no reparo das máquinas em um tempo T; ζ_{mo} representa o custo com mão de obra em um tempo T; ζ_{room} representa o montante gasto com o espaço físico para a infraestrutura de TI em um tempo T.

Estamos considerando que o tempo médio de substituição da infraestrutura de TI e equipamentos de refrigeração é de cinco anos (SCHMIDT, 2015). Assim, é necessário adicionar o valor de custo de aquisição do equipamento do sistema (σ_s), obtido a partir da Equação 5.12, aos custos totais no primeiro ano, e cinco anos depois.

5.2 Modelos de disponibilidade e confiabilidade considerando envelhecimento e rejuvenescimento de *software*

Planejar uma infraestrutura computacional significa considerar falhas de componentes que podem influenciar no funcionamento do sistema e na qualidade do serviço oferecido, bem como quais medidas podem ser tomadas para evitar ou minimizar os efeitos de tais falhas. Sendo assim, devemos considerar falhas relacionadas ao *hardware* e também relacionadas ao *software*, como é o caso do envelhecimento de *software* e de ações proativas de rejuvenescimento.

Com isso, é muito importante investigar os efeitos causados pelo envelhecimento de *software* em um ambiente de *mobile cloud computing*, pois diversos componentes da infraestrutura são compostos por *software*. Nos dispositivos móveis, a escassez de recursos é comumente

afetada pelo envelhecimento, a exemplo da memória RAM. Especialmente sistemas implantados na nuvem, os quais podem ficar longos períodos de tempo ativos, facilitando o acúmulo de pequenos erros que podem caracterizar a ocorrência do envelhecimento, podendo influenciar na disponibilidade do serviço, mesmo com a existência de recursos em abundância.

Esta subseção tem por objetivo principal descrever os modelos que representam os efeitos do envelhecimento de *software* e as diferentes ações de rejuvenescimento que podem ser adotadas para amenizar os efeitos causados pelo envelhecimento. Maiores informações sobre os experimentos realizados para coletar alguns valores de entrada dos modelos, técnicas de monitoramento utilizadas e estratégias adotadas para acelerar o surgimento dos efeitos do envelhecimento no ambiente em estudo são descritas em maiores detalhes no Apêndice A .

Modelos

Como o propósito principal desse estudo de caso é identificar o impacto dos efeitos do envelhecimento de *software* na dependabilidade de infraestruturas de *mobile cloud*, e assim poder efetuar um melhor planejamento do sistema, iremos considerar uma arquitetura de nuvem somente com os componentes básicos para o funcionamento de um sistema de saúde.

A arquitetura do sistema adotada é a mesma representada pela Figura 5.1, onde teremos ao menos um dispositivo móvel que utiliza os recursos computacionais de uma infraestrutura de nuvem, cuja troca de informações se dá através de comunicação móvel. Consequentemente, o modelo RBD da Figura 5.3 e a Equação 5.1 para calcular a disponibilidade do sistema serão mantidos, e somente os seus respectivos submodelos é que serão alterados para representar adequadamente as novas especificidades que compreendem esse estudo.

O lado do dispositivo móvel manteve as mesmas características apresentadas anteriormente. Portanto, a disponibilidade para os cenários sem envelhecimento de *software* é calculada através da Equação 5.2.

O lado da infraestrutura de nuvem foi dividido em dois componentes principais: *front-end* (FE) e o Gerente dos Nós (GN). (ver figura Figura 5.6). Portanto, a métrica de disponibilidade é calculado através da Equação 5.21.

$$A_s = A_{FE} \times (1 - (1 - A_{GN\_01}) \times (1 - A_{GN\_02})) \tag{5.21}$$

Por se tratar de uma nuvem com capacidade básica, o *front-end* abrange os principais componentes de gerenciamento da infraestrutura. O submodelo *FE* possui oito componentes: *HW*, *SO*, *GNV*, *GC*, *GAA*, *GAB*, *NAS* e *DB*. HW representa o *hardware*; SO é o sistema operacional instalado na máquina; O Gerente da Nuvem (GNV) se refere ao *webservice* responsável pelo gerenciamento de toda a infraestrutura de nuvem; O Gerente do *Cluster* (GC) é o *front-end* do *cluster*, e gerencia um conjunto de nós. GAA é o gerente de armazenamento baseado em arquivo, que representa um serviço de armazenamento de dados baseado em arquivo; GAB significa gerente de armazenamento baseado em bloco, que fornece armazenamento para uso

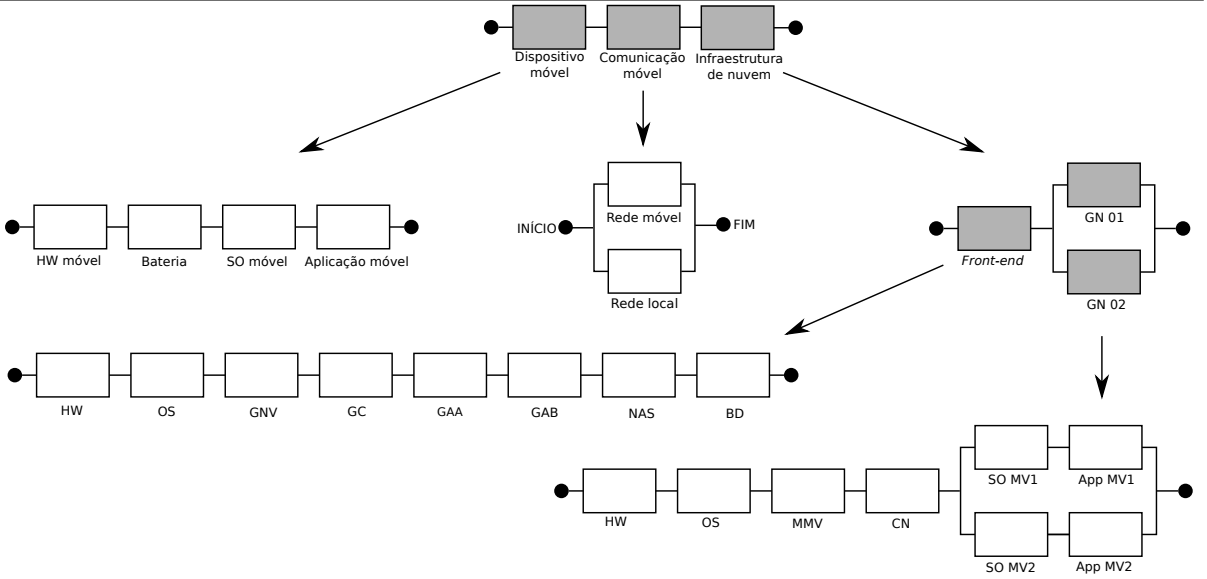


Figura 5.6: Modelo RBD para a infraestrutura de *mobile cloud*

das instâncias de máquinas virtuais; *Network-Attached Storage* (NAS) é um dispositivo que proporciona o armazenamento de dados através da rede para os gerentes de armazenamento da nuvem; e, finalmente, o BD se refere ao Sistema de Gerenciamento de Banco de Dados (SGBD).

A disponibilidade do submodelo FE é calculada através da Equação 5.22.

$$A_{fe} = A_{hw} \times A_{so} \times A_{gnv} \times A_{gc} \times A_{gaa} \times A_{gab} \times A_{nas} \times A_{db} \tag{5.22}$$

Já o submodelo GN também manteve as mesmas características descritas anteriormente. Consequentemente, sua disponibilidade pode ser calculada através da Equação 5.7.

Para os cenários com envelhecimento de *software*, a disponibilidade pode ser calculada adicionando um bloco RBD aos respectivos componentes. Por exemplo, para o lado do dispositivo móvel, a disponibilidade geral é calculada através da Equação 5.2. Mas, para os casos em que há o envelhecimento de *software*, iremos adicionar uma nova métrica chamada de *Aging Factor* (AF), cuja disponibilidade é calculada de maneira similar aos demais componentes, só que adotando o MTTARF, que representa o tempo médio de falha relacionada ao envelhecimento. Portanto, a disponibilidade é calculada através da Equação 5.23 para os cenários com envelhecimento de *software* no dispositivo móvel.

$$A_{dm} = A_{hw\_movel} \times A_{bateria} \times A_{so\_movel} \times A_{app\_movel \times A_{AF}} \tag{5.23}$$

onde os componentes A_{hw_movel}, $A_{bateria}$, A_{so_movel} e A_{app_movel} possuem os mesmos significados da Equação 5.2, e o $A_{AgingFactor}$ é calculado através da Equação $A_{AF} = \frac{\lambda_{mttarf}}{\lambda_{mttarf}+\mu_{mttarf}}$.

Para os cenários em que há a execução de alguma ação proativa de mitigação das falhas relacionadas ao envelhecimento de *software*, nós iremos adotar o modelo de Markov da Figura 5.7, que é baseado no modelo apresentado em (VAIDYANATHAN; TRIVEDI, 2005). Esse modelo considera que eventualmente o sistema pode ficar indisponível devido à ocorrência de uma falha

relacionada ao envelhecimento (λ_{mttarf}), podendo ser recuperado depois de um certo tempo (μ_{mttarf}); e que tal falha pode ser mitigada através de uma ação proativa de rejuvenescimento (λ_{mttra}), gerando um pequeno tempo de indisponibilidade controlada devido à execução da ação de rejuvenescimento (μ_{mttra}). A taxa P_{mru} a representa a probabilidade do mecanismo de revenescimento funcionar de maneira adequada, e P_{mrd} representa a probabilidade de falha desse mecanismo.

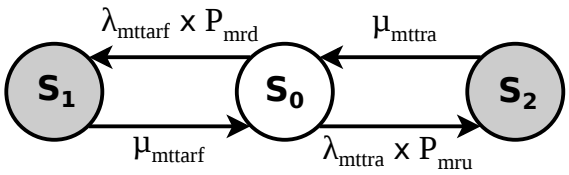


Figura 5.7: Modelo CTMC para *software* com política de rejuvenescimento

O *software* está funcionando (UP) no estado S_0. A ocorrência de uma falha causada pelo envelhecimento de *software* é representada pela transição de S_0 para S_1. Essa transição ocorre com taxa $\lambda_{mttarf} \times P_{mrd}$. No estado S_1, o *software* não está funcionando (DOWN), então ele não pode ser acessado pelos usuários. Quando o sistema estiver no estado S_1, o modelo pode voltar para o estado S_0 depois de uma recuperação reativa (μ_{mttarf}), ou seja, volta a funcionar após ocorrer uma falha ocasionada pelo envelhecimento de *software*. Por outro lado, a execução com sucesso da ação de rejuvenescimento leva o modelo do estado S_0 para o estado S_2. Este estado representa que o *software* está sendo rejuvenescido. A transição do estado S_0 para S_2 ocorre com taxa $\lambda_{mttra} \times P_{mru}$. A transição do estado S_2 para o estado S_0 representa a conclusão de uma ação proativa, e possui taxa μ_{mttra}. O *software* é considerado indisponível no estados S_1 e S_2.

A disponibilidade desse modelo pode ser calculada através da fórmula fechada apresentada na Equação 5.24.

$$A_{rej} = \frac{\mu_{mttarf}\mu_{mttra}}{P_{mru}\lambda_{mttra}\mu_{mttarf} + (P_{mrd}\lambda_{mttarf}\mu_{mttarf})\mu_{mttra}} \tag{5.24}$$

onde λ_{mttarf} é a taxa média da falha relacionada ao envelhecimento de *software*; λ_{mttra} representa a taxa média da ação de rejuvenescimento; P_{mru} refere-se à probabilidade do mecanismo de rejuvenescimento ser executado com sucesso, acionando adequadamente uma ação de rejuvenescimento; P_{mrd} refere-se à probabilidade de insucesso do rejuvenescimento, que é a falha na execução da ação de rejuvenescimento.

5.3 Modelos para avaliação do tempo de vida da bateria em dispositivos móveis

O estudo da dependabilidade em sistemas de *mobile cloud computing* aponta para a bateria do dispositivo móvel como sendo um dos componentes mais importantes. Este fato se dá não somente pela sua influência na dependabilidade em curto prazo, mas também a longo

prazo. A descarga rápida influencia no consumo dos ciclos de vida da bateria, podendo levar à substituição prematura da bateria, já que alguns dispositivos móveis possuem bateria acoplada ao restante do *hardware*.

Para facilitar o entendimento do funcionamento dos modelos, vamos aplicá-lo em uma arquitetura de um sistema de saúde móvel padrão. No entanto, ele poderia ser aplicado a qualquer outro cenário em que há a necessidade de comunicação de um dispositivo móvel com um servidor remoto usando redes sem fio.

Uma arquitetura de uma *mobile cloud computing* é composta por três componentes principais: o paciente (dispositivo móvel), a conectividade (redes de comunicação), e a infraestrutura de computação (nuvem). No entanto, como o objetivo deste estudo é identificar a vida útil da bateria, portanto, só vamos considerar os fatores que afetam diretamente a descarga da bateria, como os protocolos de mensagens e os diferentes tipos de comunicação de rede sem fio. Portanto, não iremos considerar a infraestrutura computacional para os modelos propostos nesse estudo. A Figura 5.8 apresenta uma visão geral da arquitetura de um sistema móvel, com os principais componentes de interesse.

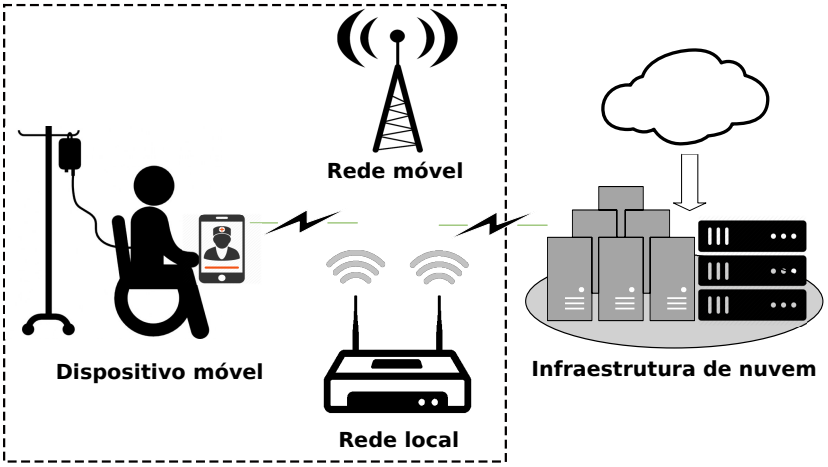


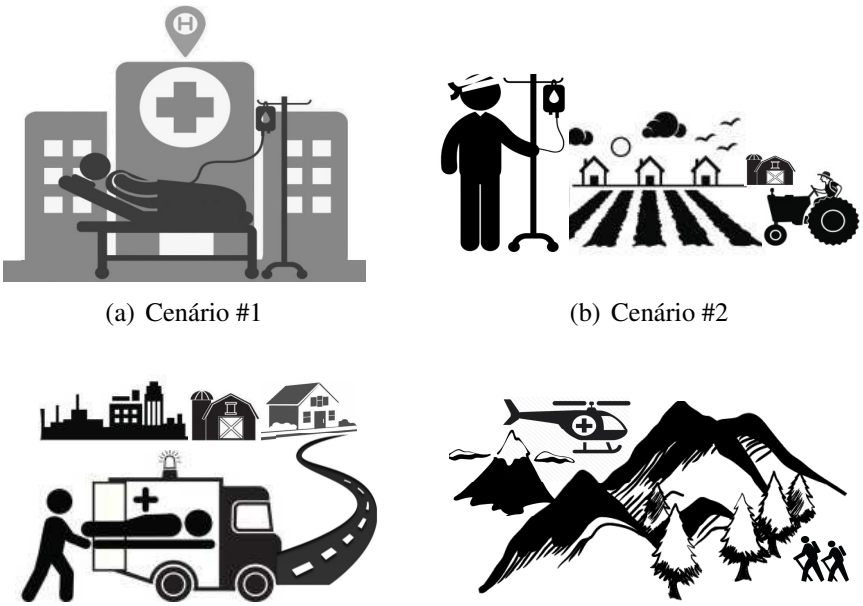
Figura 5.8: Visão geral da arquitetura do sistema móvel

Nesse cenário, o *paciente* é uma pessoa que precisa de algum tipo de atenção médica, e que tem algum tipo de dispositivo de monitoramento de sinais vitais. Tal dispositivo se conecta a um *smartphone*, e o *smartphone* é responsável para se comunicar com o servidor através de qualquer rede de comunicação disponível. Para *conectividade*, estamos levando em conta que tal dispositivo móvel é capaz de se conectar através de uma *rede local*[3] ou uma *rede móvel*[4]. No entanto, para que a conectividade seja possível, uma das duas redes deve estar disponível, ou seja, o roteador (rede local) ou a torre de comunicação (rede móvel) deve estar funcionando. Além disso, o *smartphone* também deve estar em uma área de cobertura e sem qualquer obstáculo físico/magnético. Finalmente, a *infraestrutura de computação* é responsável por armazenar as informações enviadas através da rede, e processar os pedidos recebidos. Para o

[3]Para simplificar, vamos nos referir à rede local sem fio (WLAN) apenas como *rede local*.
[4]Redes 2G, 3G, 4G, etc.

nosso propósito, estamos levando em conta que esta infraestrutura pode ser um único servidor ou uma infraestrutura mais complexa, como uma nuvem privada.

Considerando que os mesmos componentes representados na Figura 5.8 serão mantidos, então somente a probabilidade de conectividade é que deverá mudar de acordo com os diferentes cenários. Portanto, vamos considerar quatro cenários diferentes, com conectividades distintas. No entanto, vamos assumir que uma rede local tem maior prioridade que a rede móvel. A Figura 5.9 mostra diferentes cenários da vida real.



(a) Cenário #1 (b) Cenário #2

(c) Cenário #3 (d) Cenário #4

Figura 5.9: Cenários com diferentes tipos de conectividade

A Figura 5.9(a) mostra o cenário #1, onde o paciente possui boa conectividade em ambas as redes de comunicação. Por exemplo, um paciente em um hospital sob cuidados médicos que necessita de um acompanhamento constante de seus sinais vitais.

O cenário #2 (ver Figura 5.9(b)) considera um cenário com boa conectividade para a rede local, mas com pouca conectividade para a rede móvel. Por exemplo, um paciente em recuperação que vive em uma área rural, mas ainda precisa de algum tipo de monitoramento dos sinais vitais.

A Figura 5.9(c) mostra o cenário #3, que é o oposto do cenário #2, ou seja, representa um cenário com boa conectividade para a rede móvel, mas com fraca conectividade para a rede local. Por exemplo, um cenário dentro da cobertura da rede móvel, mas longe da área de cobertura de uma rede local, tal como numa ambulância.

Finalmente, o cenário #4 (Figura 5.9(d)) representa o oposto do cenário #1, ou seja, pouca conectividade para ambas as redes de comunicação. Por exemplo, um paciente num local remoto sem área de cobertura de ambas as redes, como uma montanha ou vale.

Os modelos propostos visam representar o comportamento da descarga da bateria durante a utilização de um sistema móvel. Especialmente o tempo de descarga de cada ciclo, considerando

diferentes protocolos de mensagens e cenários com conectividades distintas.

Devido à sua eficiência computacional, adotamos modelos CTMC para representar a bateria e as condições ambientais. Adotamos uma abordagem de modelagem hierárquica heterogênea, combinando diferentes modelos CTMC.

O sistema móvel avaliado nesta tese é ilustrado na Figura 5.8. Para representar um ambiente como esse e estimar o tempo de vida da bateria, foram criados alguns modelos analíticos.

Para calcular a vida útil total da bateria (γ), adotamos a Equação 5.25:

$$\gamma = (\phi + \mu) \times \beta \tag{5.25}$$

onde ϕ é o tempo médio de descarga da bateria. μ representa o tempo médio para recarregar uma bateria em sua capacidade máxima. E β é o número médio de ciclos de vida da bateria.

O tempo médio de um ciclo de vida da bateria (ϕ) é representado pelo modelo CTMC apresentado na Figura 5.10.

Figura 5.10: Modelo CTMC de descarga da bateria

A fórmula fechada para o tempo médio de descarga de um ciclo de vida da bateria (ϕ) é expressada pela Equação 5.26.

$$\phi = \frac{\kappa}{\delta} \tag{5.26}$$

onde κ representa o número de estágios do modelo. Neste caso, $\kappa = 10$. E δ representa a taxa de transição para cada estado, que é calculado pela Equação 5.27.

Este modelo de estado absorvente segue uma distribuição Erlang (TRIVEDI, 2002; CLOTH; JONGERDEN; HAVERKORT, 2007). Todas as transições possuem uma taxa δ, que é calculada pela Equação 5.27.

$$\delta = \zeta + \rho + \omega + \eta \tag{5.27}$$

onde ζ representa o consumo de bateria da tela do *smartphone*. ρ se refere à energia gasta com processamento de aplicações e processos do sistema operacional; Componentes como CPU (ρ_{cpu}) e memória RAM (ρ_{ram}) podem ser considerados separadamente. ω representa a taxa de descarga dos protocolos de comunicação (veja equação 5.28). Finalmente, η se refere à soma dos fatores complementares ou de componentes auxiliares ($\eta = \sum_{i=1}^{n} \eta_i$) que podem consumir recursos da bateria, como temperatura, GPS, acelerômetro, *bluetooth*, cartão SD, etc.

$$\omega = (\lambda_{rl} \times P_{rl}) + (\lambda_{rm} \times P_{rm}) + (\lambda_{none} \times P_{none}) \tag{5.28}$$

onde λ_{rl} representa a taxa de descarga para cada protocolo de mensagem em uma rede local. P_{rl} é a probabilidade de conectividade da rede local, isto é, a rede local tem de estar funcionando e em uma área de cobertura. λ_{rm} representa a taxa de descarga para cada protocolo de mensagem em uma rede móvel. P_{rm} refere-se à probabilidade de conectividade da rede móvel. Similar aos demais, λ_{none} e P_{none} representa a taxa de descarga de um dispositivo móvel em um cenário em que ele não está conectado a qualquer rede e sua probabilidade de conectividade, respectivamente. A soma de P_{rl}, P_{rm}, e P_{none} deve ser igual a 1.

Os parâmetros de λ_{rl}, λ_{rm}, e λ_{none} serão obtidos a partir de estudos experimentais, os quais estão descritos em detalhes na Subseção B. No entanto, para a conectividade (P_{rl}, P_{rm}, e P_{none}), tais métricas são obtidas através do modelo *Markov Reward Models* (MRM) mostrado na Figura 5.11.

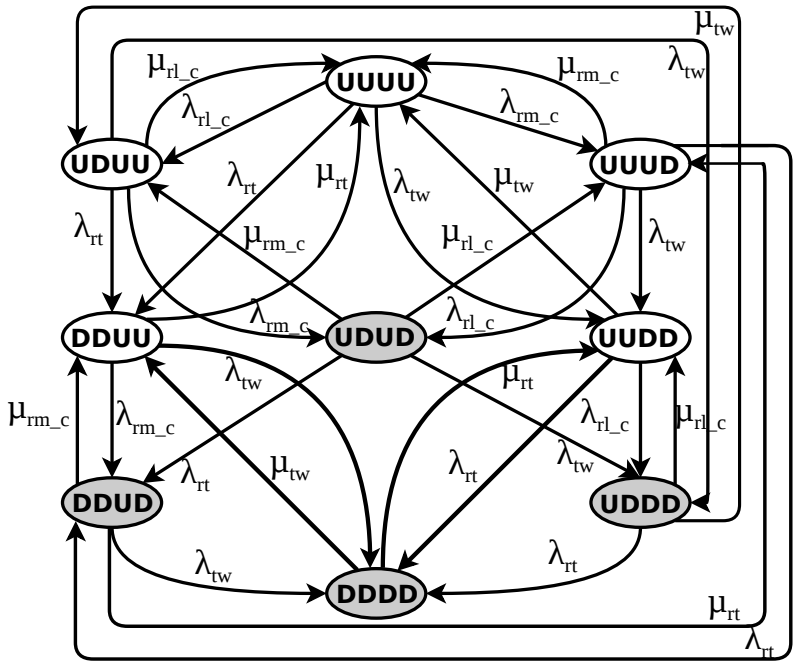


Figura 5.11: Modelo CTMC para conectividade da rede

Esta CTMC modela o comportamento do ambiente, considerando as falhas físicas e problemas de cobertura para cada cenário. Consideramos que o tempo médio até a falha (MTTF) e tempo médio de reparo (MTTR) de cada componente são exponencialmente distribuídos.

Cada estado é representado por quatro letras que podem ser *U* (UP), or *D* (down). A primeira letra indica se o dispotivo móvel está na área de cobertura da rede local (*U*), ou não (*D*). A segunda letra denota se há um problema com a infraestrutura de rede local, como um roteador, (*D*), ou não (*U*). A terceira letra mostra se o paciente está na área de cobertura da rede móvel (*U*), ou não (*D*). Finalmente, a quarta letra informa se há algum problema na infraestrutura da rede móvel (*D*), ou não (*U*).

O modelo MRM da Figura 5.11 possui nove estados: $UUUU$, $UDUU$, $UUUD$, $DDUU$, $UDUD$, $UUDD$, $DDUD$, $UDDD$, e $DDDD$. O sistema está disponível nos estados $UUUU$, $UDUU$, $UUUD$, $DDUU$, e $UUDD$ (estados brancos), uma vez que existe conectividade nesses

estados. E está indisponível nos estados $UDUD$, $DDUD$, $UDDD$, e $DDDD$ (estados cinzas), já que não há conectividade nesses estados.

Portanto, considerando que a rede local tem prioridade sobre a rede móvel, nós organizamos os estados da seguinte forma: $\lambda_{rl} = UUUU, UUUD, UUDD$; $\lambda_{rm} = UDUU, DDUU$; $\lambda_{none} = UDUD, DDUD, UDDD, DDDD$. Assim, estas métricas são obtidas pela soma da probabilidade de cada estado. A fórmula fechada para obter os resultados da conectividade é expressada pela Equação 5.29.

$$\lambda_{cm} = 1 - \frac{\sigma_1}{\sigma_2} - \frac{\sigma_3(\sigma_4 + \mu_{rt})}{\sigma_2(\sigma_5\sigma_4 + \sigma_6\mu_{rt})} - \frac{\sigma_7}{\sigma_8} - \frac{\sigma_9\tau_6}{\tau_7(\tau_8 + \tau_9)\upsilon_1\upsilon_2} \tag{5.29}$$

onde,

$\sigma_1 = \lambda_{tw}\lambda_{rt}$,
$\sigma_2 = (\lambda_{tw} + \mu_{tw})(\lambda_{rt} + \mu_{rt})$,
$\sigma_3 = \lambda_{rm\_c}\lambda_{rt}\mu_{tw}$,
$\sigma_4 = \lambda_{tw} + \lambda_{rm\_c} + \lambda_{rt} + \mu_{rm\_c}$,
$\sigma_5 = \lambda_{tw} + \lambda_{rm\_c} + \mu_{tw}$,
$\sigma_6 = \lambda_{tw} + \lambda_{rm\_c} + \mu_{rm\_c}$,
$\sigma_7 = \lambda_{tw}\lambda_{rl\_c}\mu_{rt}$,
$\sigma_8 = (\lambda_{tw} + \mu_{tw})(\lambda_{rt} + \mu_{rt})(\lambda_{rt} + \lambda_{rl\_c} + \mu_{rl\_c})$,
$\sigma_9 = \lambda_{rm\_c}\lambda_{rl\_c}\mu_{tw}\mu_{rt}$,
$\tau_1 = \lambda_{tw}^2 + \lambda_{rm\_c}^2 + \lambda_{rt}^2 + \lambda_{rt}\lambda_{rl\_c} + 2\lambda_{rt}\mu_{tw}$,
$\tau_2 = \lambda_{rl\_c}\mu_{tw} + \mu_{tw}\mu_{rm\_c} + \lambda_{rt}\mu_{rt} + \lambda_{rl\_c}\mu_{rt} + \mu_{rm\_c}\mu_{rt}$,
$\tau_3 = (\lambda_{rt} + \mu_{rm\_c} + \mu_{rt})\mu_{rl\_c}$,
$\tau_4 = \lambda_{rm\_c}(2\lambda_{rt} + \lambda_{rl\_c} + \mu_{tw} + \mu_{rm\_c} + \mu_{rt} + \mu_{rl\_c})$,
$\tau_5 = \lambda_{tw}(2\lambda_{rm\_c} + 2\lambda_{rt} + \lambda_{rl\_c} + \mu_{tw} + \mu_{rm\_c} + \mu_{rt} + \mu_{rl\_c})$,
$\tau_6 = (\tau_1 + \tau_2 + \tau_3 + \tau_4 + \tau_5)$,
$\tau_7 = (\lambda_{tw} + \mu_{tw})(\lambda_{rt} + \mu_{rt})$,
$\tau_8 = (\lambda_{tw} + \lambda_{rm\_c} + \mu_{tw})(\lambda_{tw} + \lambda_{rm\_c} + \lambda_{rt} + \mu_{rm\_c})$,
$\tau_9 = (\lambda_{tw} + \lambda_{rm\_c} + \mu_{rm\_c})\mu_{rt}$,
$\upsilon_1 = (\lambda_{rt} + \lambda_{rl\_c} + \mu_{rl\_c})$,
$\upsilon_2 = (\lambda_{tw} + \lambda_{rm\_c} + \lambda_{rt} + \lambda_{rl\_c} + \mu_{rm\_c} + \mu_{rl\_c})$.

No estado inicial $UUUU$, o roteador da rede local está funcionando e o paciente está dentro da área de cobertura da rede local, e a torre de comunicação da rede móvel também está funcionando e o paciente também está dentro da área de cobertura da rede móvel. Quando o paciente já não está na área de cobertura da rede local (λ_{rl_c}), o sistema vai para o estado $UDUU$, onde o roteador ainda está funcionando, mas não há mais conectividade para a rede local; neste estado, o paciente pode voltar para a área de cobertura da rede local (μ_{rl_c}), retornando o sistema para o estado $UUUU$. Mais uma vez no estado inicial, quando o roteador falhar (λ_{rt}), o sistema

vai para o estado $DDUU$, o que significa que a rede móvel vai ser usada, uma vez que já não há uma área de cobertura da rede local. Neste estado, o roteador pode ser reparado (μ_{rt}), retornando o sistema para o estado $UUUU$, também restaurando a área de cobertura.

Ainda no estado inicial, um comportamento semelhante pode ser observado na rede móvel. Quando o paciente já não está na área de cobertura (λ_{rm_c}), o sistema vai para o estado $UUUD$, onde a torre de comunicação está funcionando, mas não há mais conectividade para a rede móvel. Neste novo estado, o paciente pode voltar para a área de cobertura da rede móvel (μ_{rm_c}), trazendo o sistema para o estado $UUUU$. Mais uma vez no estado inicial, quando a torre de comunicação falhar (μ_{rm_c}), o sistema vai para o estado $UUDD$, o que significa que a rede local será utilizada, uma vez que já não há uma área de cobertura da rede móvel. Neste estado, a torre de comunicação pode ser reparada (μ_{tw}), trazendo o sistema mais uma vez para o estado $UUUU$.

Estando nos estados intermediários $UDUU$ e $UUUD$, o sistema pode falhar e ainda continuar funcionando. No estado $UDUU$, se o roteador falhar (λ_{rt}), o sistema vai para o estado $DDUU$. No estado $UUUD$, se a torre de comunicação falhar (λ_{tw}), o sistema vai para o estado $UUDD$. Assim, em ambos os novos estados ainda haverá conectividade. No entanto, o estado $UDUU$ pode alcançar o estado de não-conectividade $UDUD$ se o paciente se mover para uma área sem cobertura da rede móvel (λ_{rm_c}), ou o estado $UDUU$ pode alcançar o outro estado de não-conectividade $UDDD$ através da falha da torre de comunicação (λ_{tw}); e o estado $UUUD$ pode alcançar o estado de não-conectividade $UDUD$ se o paciente se mover para uma área sem cobertura da rede local (λ_{rl_c}), ou o estado $UUUD$ pode alcançar outro estado sem conectividade $DDUD$ através da falha do roteador (λ_{rt}).

Considerando os estados intermediários onde há conectividade, o estado de falha total $DDDD$ pode ser alcançado a partir do estado $DDUU$, através da falha da torre de comunicação (λ_{tw}); e a partir do estado $UUDD$, através da falha do roteador (λ_{rt}). Agora, considerando os estados onde não há conectividade, o estado de falha total $DDDD$ pode ser alcançado a partir do estado $DDUD$, através da falha da torre de comunicação (λ_{tw}); e a partir do estado $UDDD$, através da falha do roteador (λ_{rt}).

Mesmo se um cenário alcança um valor alto de vida útil não significa que o serviço será fornecido adequadamente. Portanto, também é necessário calcular a disponibilidade de todo o sistema. Para isso, a arquitetura geral de computação móvel é representada por um modelo RBD representado na Figura 5.12. Cada subsistema tem suas métricas de disponibilidade computadas através dos correspondentes submodelos. Consideramos que o tempo médio até a falha (MTTF) e tempo médio de reparo (MTTR) de cada componente são exponencialmente distribuídos.

A fórmula fechada para calcular a disponibilidade do sistema é expressada pela Equação 5.30. Cada componente A_X na equação é computado a partir da avaliação do respectivo submodelo (A_{dm}, e A_{cm}), que também podem ser feitos através de equações de fórmula fechada, se for possível obtê-las, ou através de uma solução numérica.

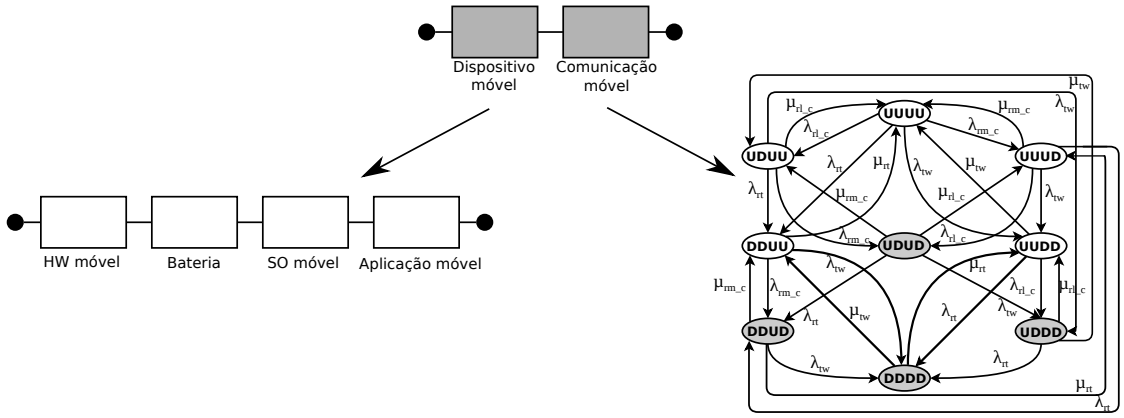


Figura 5.12: Modelo RBD para a computação móvel

$$A_s = A_{dm} \times A_{cm} \tag{5.30}$$

A disponibilidade do dispositivo móvel é calculada a partir da Equação 5.2. O subsistema da comunicação móvel também é representado pelo modelo ilustrado na Figura 5.11, e os resultados de disponibilidade são obtidos a partir da Equação 5.29.

5.4 Modelo de entrega de mensagens em sistemas de MCC

O estudo da dependabilidade em sistemas móveis necessita não somente de melhorias arquiteturais que influenciam na obtenção de altos níveis de disponibilidade e confiabilidade do sistema, mas também é necessário identificar qual o impacto que tais falhas/melhorias podem ocasionar no tempo médio que uma mensagem de comunicação demora para ser entregue ao seu destinatário. Para alguns tipos de sistemas, esse atraso pode não implicar impactos negativos. Em outros, pode impactar na satisfação do usuário, como um serviço de GPS que demora alguns segundos a mais para apresentar o resultados. No entanto, para outros, os resultados podem ser catastróficos, a exemplo de sistemas *mHealth*, onde a demora na entrega de uma mensagem coletada de um paciente que deverá ser entregue à equipe médica pode acarretar em danos irreparáveis, inclusive a morte de pacientes.

A Figura 5.13 apresenta a arquitetura que será considerada nesse cenário, onde o **paciente** tem os seus sinais vitais monitorados e enviados para uma infraestrutura de **computação em nuvem** através de uma conexão de rede fixa. A infraestrutura de nuvem é responsável por armazenar e processar os dados, enviando as informações críticas para o **dispositivo móvel** da equipe médica através das redes local ou móvel.

A Figura 5.14 mostra um modelo de alto nível que representa o caminho percorrido por mensagem, uma vez que foi coletada do paciente até chegar ao dispositivo móvel, e servirá para identificar o tempo médio e probabilidade da mesma ser entregue.

O modelo CTMC da Figura 5.14 se trata de um modelo de estado absorvente de cinco estados, onde o estado S_0 representa o ponto inicial onde a mensagem é criada, e o estado S_3

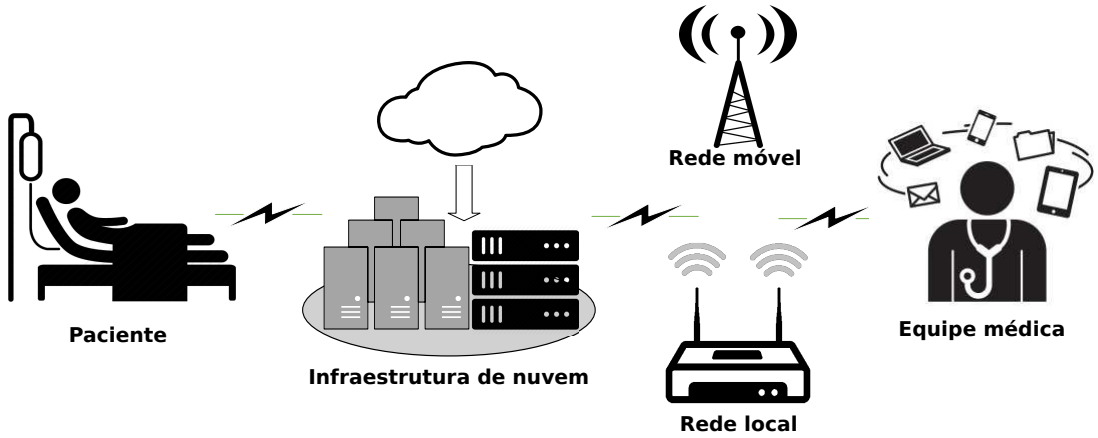


Figura 5.13: Visão geral de uma *mobile cloud* para sistemas *mHealth*

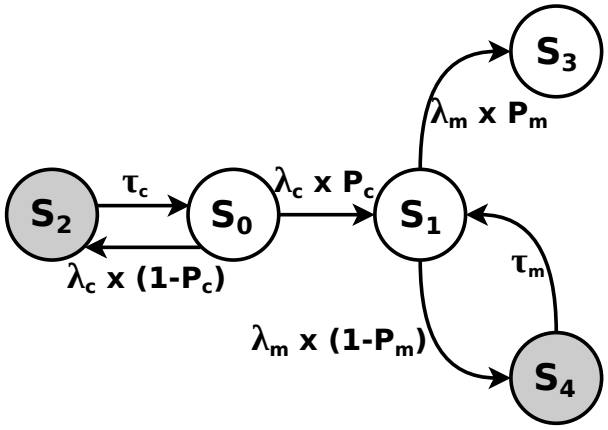


Figura 5.14: Modelo de entrega de mensagens

indica que a mensagem foi entregue com sucesso ao destinatário. Após sair do estado S_0, a mensagem deve ser enviada para a nuvem através da taxa de envio λ_c, com probabilidade P_c, que representa a probabilidade da infraestrutura de nuvem e da comunicação móvel estarem disponíveis. No entanto, caso a infraestrutura de nuvem esteja indisponível $(1-P_c)$, a mensagem será encaminhada para o estado S_2. No estado S_2, o sistema irá aguardar um *timeout* τ_c para reenviar a mesma mensagem novamente, retornando ao estado S_0, para após seguir para o estado S_1. O estado S_1 significa que a mensagem foi entregue com sucesso à infraestrutura de nuvem. Do estado S_1, a mensagem deverá ser entregue ao estado S_3 através da taxa de envio para o dispositivo móvel (λ_m), com probabilidade P_m, que representa a probabilidade do dispositivo móvel e da comunicação móvel estarem disponíveis. Caso o dispositivo móvel esteja indisponível $(1-P_m)$, a mensagem será entregue ao estado S_4, onde deverá aguardar um *timeout* τ_m para reenviar a mensagem para o dispositivo móvel, voltando para o estado S_1.

O tempo médio médio gasto por uma mensagem para percorrer todo esse caminho é obtido através da Equação 5.31:

$$TM = \frac{\frac{1}{\lambda_c}+\frac{1}{\tau_c}}{P_c} + \frac{\frac{1}{\lambda_m}+\frac{1}{\tau_m}}{P_m} - \frac{\tau_c+\tau_m}{\tau_c\tau_m} \tag{5.31}$$

onde λ_c representa a taxa de envio da mensagem para a nuvem. O parâmetro P_c refere-se à probabilidade da infraestrutura de nuvem estar disponível para receber a mensagem enviada pelo dispositivo móvel, a qual será obtida através da Equação 5.4. O parâmetro λ_m representa a taxa de envio para o dispositivo móvel, onde $P_m = A_{cm} + A_{dm}$; o parâmetro A_{cm} se refere à disponibilidade da comunicação móvel obtida a partir da Equação 5.29; A_{dm} refere-se à disponibilidade do dispositivo móvel, obtida através da Equação 5.2.

Maiores detalhes sobre os modelos do dispositivo móvel, da comunicação móvel (Figura 5.11) e da infraestrutura de computação em nuvem, podem ser encontrados nas Seções 5.1 e 5.3.

5.5 Considerações finais

Os modelos propostos neste capítulo visam auxiliar no planejamento de infraestruturas de MCC com foco na melhoria dos níveis de disponibilidade e confiabilidade do sistema. Todos os modelos foram desenvolvidos a partir dos formalismos RBD e CTMC. A partir desses modelos é possível efetuar a avaliação do sistema e obter resultados que podem auxiliar na tomada de decisão. Vale ressaltar que os métodos apresentados aqui foram aplicados com sucesso em vários estudos de caso, demonstrados no Capítulo 6.

6

Estudos de caso

Não me entrego sem lutar. Tenho ainda coração. Não aprendi a me render.
Que caia o inimigo então.

—RENATO RUSSO

Este capítulo apresenta quatro cenários com diversos estudos de caso que demonstram a metodologia utilizada para apoiar o planejamento de infraestruturas de *mobile cloud computing*: (i) a avaliação de disponibilidade e confiabilidade de um ambiente de *mobile cloud* considerando a arquitetura e o custo; (ii) a avaliação da dependabilidade de uma MCC considerando a existência do envelhecimento de *software* nos principais componentes do sistema e com o acionamento de estratégias de rejuvenescimento; (iii) a avaliação do tempo de vida da bateria em dispositivos móveis considerando diferentes tipos de protocolos de comunicação sob condições distintas de conectividade. (iv) identificação do tempo médio de entrega de mensagens entre dispostivos móveis considerando falhas e reparos nos principais componentes da infraestrutura. Os resultados obtidos nesses estudos de caso fornecem evidência sobre a utilidade e eficácia desta abordagem.

6.1 Avaliação de disponibilidade e confiabilidade *vs* custo de implantação e provisionamento

Estudos de casos distintos são apresentados nesta seção para avaliar a infraestrutura de MCC de acordo com os modelos apresentados na Seção 5.1. A primeira parte da análise tem como objetivo avaliar a disponibilidade e a confiabilidade de todo o sistema de MCC, considerando cenários com baterias redundantes (reposição) e sem reposição. A segunda parte da nossa análise está focada em diferentes esquemas de redundância para melhorar a disponibilidade da infraestrutura de MCC.

O principal objetivo desse estudo de caso é estimar a disponibilidade e a confiabilidade do sistema, considerando diferentes parâmetros do sistema, e identificar o custo de instalação e manutenção de um ambiente de nuvem privada.

6.1.1 Parâmetros de entrada

A Tabela 6.1 apresenta os parâmetros de entrada para o modelo do dispositivo móvel. O MTTF do *hardware* móvel vem da *Annual Failure Rate* (AFR) relatada em (SQUARETRADE, 2010), mas foi reduzida por um fator de 10. Esta redução tem como objetivo obter um valor mais realista do MTTF baseado em (SCHROEDER; GIBSON, 2007), que afirma que o tempo entre as substituições de unidades de disco rígido com falha pode ser 10 vezes menor do que o MTTF publicado pelos fabricantes. O MTTF e MTTR da bateria se referem à média dos tempos de descarga e recarga, obtidos nas especificações publicadas pelos principais fabricantes de *smartphones* (PHONEARENA, 2015). Os valores de MTTF e MTTR da aplicação são estimados em (KIM; MACHIDA; TRIVEDI, 2009).

Tabela 6.1: Parâmetros de entrada para o modelo do dispositivo móvel

Componente	**MTTF (horas)**	**MTTR (horas)**
Hardware móvel	22461,5	1,6667
Bateria	9,0	1,5
SO móvel	1440,9	0,03333
Aplicação móvel	336,7	0,01666

A Tabela 6.2 mostra os parâmetros de MTTF e MTTR para os blocos que representam a rede local e a rede móvel. Estes valores são apresentados em (D-LINK WIRELESS N150 ROUTER, 2015) e (COOPER; FARRELL, 2007), respectivamente.

Tabela 6.2: Parâmetros de entrada para as redes de comunicação

Componente	**MTTF (horas)**	**MTTR (horas)**
Rede local (roteador)	10000	1,6666
Rede móvel (torre de transmissão)	83220	12

A Tabela 6.3 descreve uma compilação dos parâmetros utilizados em todos os submodelos de nuvem: GI, GA, e GN. Estes valores foram encontrados no estudo do sistema Eucalyptus apresentado em (DANTAS et al., 2012). Uma vez que os principais módulos de *software* da nuvem são implementados como *webservices*, nós também assumimos que eles possuem características de falha e de reparo semelhantes entre si. Para o componente NAS, o seu parâmetro MTTF foi obtido através de (SCHROEDER; GIBSON, 2007), e o MTTR foi considerado o mesmo do *hardware*.

Em nosso caso, a potência de entrada (ρ_{input_i}) é a mesma para todas as máquinas, e o seu valor é 0,1295 kW (DELL, 2016a), considerando um ambiente com uma voltagem de entrada de 220 AC e 25^{o}C. Também foi adotado um custo médio de eletricidade (ξ_{power}) de US$ 0,165/kWh (ENERGY, 2013).

Tabela 6.3: Parâmetros de entrada para todos os subsistemas da nuvem

Componente	**MTTF (horas)**	**MTTR (horas)**	Subsistema
HW	8760	1,66667	Todos
SO	2893	0,25	Todos
GNV/GC	788	1	GI
GAA/GAB	788	1	GA
NAS	1000	1,66667	GA
MMV	2990	1	GN
CN	788	1	GN
SO_MV	2893	0,25	GN
App_MV	788	1	GN

6.1.2 Resultados dos modelos

As métricas de *Availability Importance* (AI) e *Reliability Importance* (RI) foram calculadas para identificar os componentes mais importantes em todo o sistema de MCC. A Tabela 6.4 mostra que a *Bateria* é o componente mais importante seguido pelo *GA* e do *GI*

Tabela 6.4: AI e RI da *mobile cloud*

Componentes	**Importância de Componentes**	
	Availability Importance	***Reliability Importance***
Bateria	1,0	1,0
GA	0,86098406	0,07648066
GI	0,85955549	0,07466698
Hardware móvel	0,85720646	0,06955773
App móvel	0,85718527	0,07461702
SO móvel	0,85716268	0,07065048
Rede móvel	0,00142614	0,00164776
Rede local	0,00012358	0,00002004
GN 01-05	3,3080E-11	0,00000127

Uma vez identificados os componentes mais importantes, é possível propor diferentes estratégias de redundância para melhorar a disponibilidade e a confiabilidade do sistema.

A análise dos modelos para a MCC incluem dois cenários: no cenário #1A (*baseline*), o dispositivo móvel é usado com uma única bateria, portanto, a descarga total da bateria acarretará na indisponibilidade do sistema e o serviço somente estará disponível após uma recarga completa. Observe que muitos dispositivos móveis podem ser usados durante a recarga, mas o elemento-chave da mobilidade do usuário é perdido, por isso esta é a razão pela qual consideramos que o sistema só está disponível depois de uma recarga completa. O cenário #1B considera a existência de uma bateria reserva, que pode substituir a bateria principal logo após a descarga total, portanto, não há necessidade de esperar a recarga da bateria para o sistema estar disponível novamente. Alguns fabricantes de *smartphones* incentivam o uso de baterias reservas com a venda de carregadores específicos para baterias sobressalentes. A comparação entre os cenários

propostos visam verificar o quanto tal comportamento melhora a disponibilidade do aplicativo no contexto da MCC.

A Tabela 6.5 mostra os resultados para as métricas dos cenários #1A e #1B. Vale destacar a diferença de mais de 1 hora no MTTR entre ambos os cenários. Esta diferença é esperada devido à presença de uma bateria reserva que evita a espera por uma recarga completa da bateria principal. Como consequência, a disponibilidade do sistema aumenta significativamente, de 85% no cenário #1A para 97% no cenário #1B. A melhoria na disponibilidade pode ser também medida através da diminuição do tempo de *downtime* anual, ou seja, a quantidade de tempo que o usuário é incapaz de acessar o serviço móvel devido a qualquer tipo de falha. O *downtime* do cenário #1A é de cerca 1.307 horas, e de 179 horas para o cenário #1B. Este resultado indica que incentivando os usuários a usarem baterias reserva é um meio de garantir o uso ininterrupto de aplicativos que usam nuvens móveis.

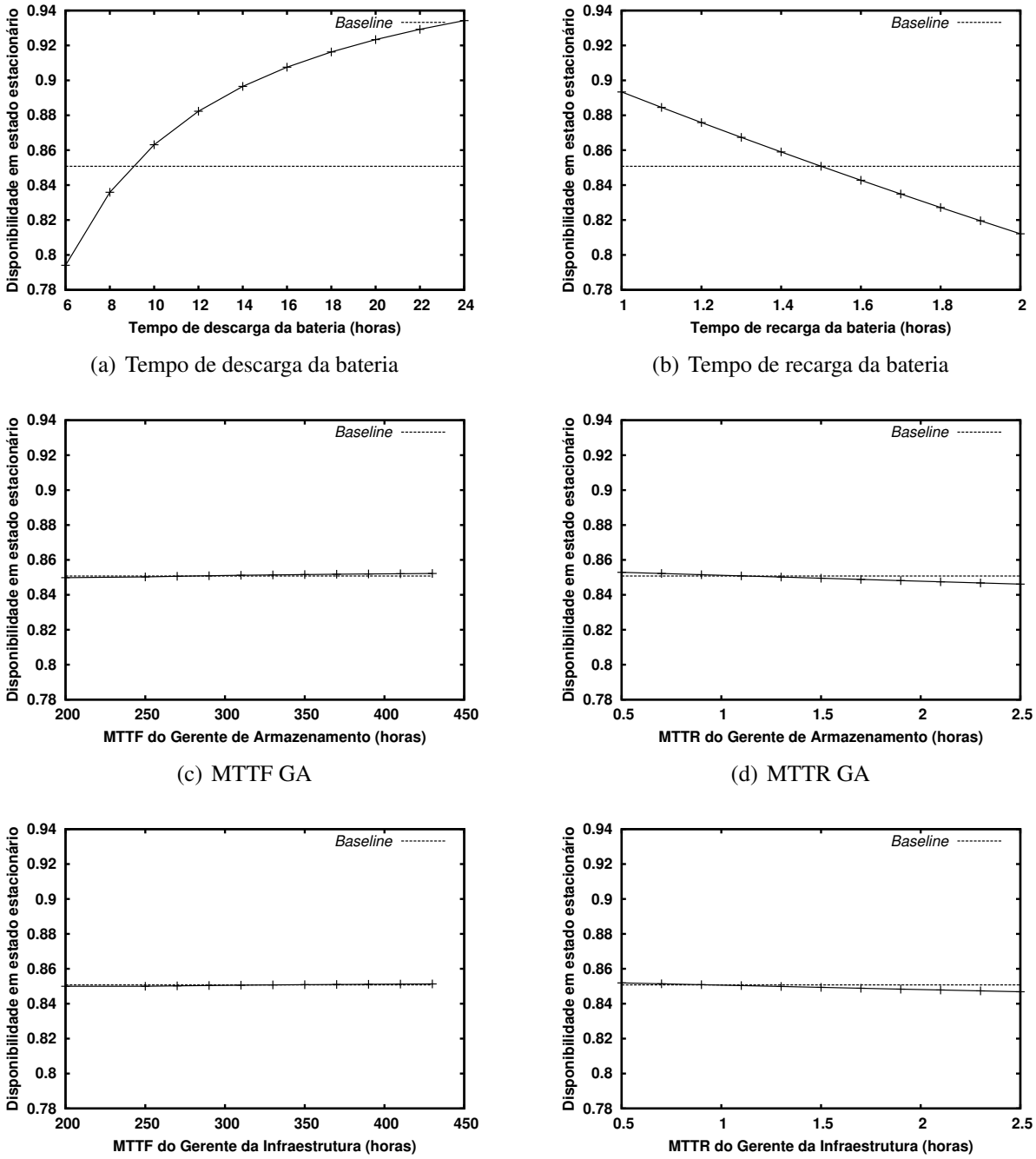
Tabela 6.5: Resultado de disponibilidade da *mobile cloud*

Métricas	**Cenários**	
	#1A	**#1B**
MTTF (horas)	8,02865	8,02865
MTTR (horas)	1,42388	0,17815
Disponibilidade (%)	85,07985	97,95074
Disponibilidade (9's)	0,82622	1,68840
Uptime (horas/ano)	7.452,99533	8.580,48495
Downtime (horas/ano)	1.307,00467	179,51505

A Figura 6.1 mostra uma análise de sensibilidade dos três componentes mais importantes para a MCC: Bateria, Gerente de Armazenamento e Gerente da Infraestrutura. Como esperado, as mudanças na descarga da bateria e os tempos de recarga são mais significativos do que os componentes GA e GI. De fato, o tempo médio de descarga da bateria usada no cenário de referência (*baseline*) considera a utilização ininterrupta do dispositivo móvel, ou seja, o pior caso.

A Figura 6.1(a) mostra que a disponibilidade também aumenta quando os valores do tempo de descarga da bateria são aumentados, mas com uma taxa de crescimento menor. Por exemplo, quando o tempo de descarga aumenta de 6 para 8 horas, a disponibilidade sobe 4 pontos percentuais, de 79,407% para 83,587%. Por outro lado, quando o valor de descarga aumenta de 22 para 24 horas, a disponibilidade aumenta de 92,924% para 93,421%, ou seja, menos da metade de um ponto percentual. Figura 6.1(b) confirma que as baterias com um menor tempo de recarga têm melhores resultados de disponibilidade. A variação paramétrica dos valores de MTTF dos componentes GA e GI (Figuras 6.1(c) e 6.1(e), respectivamente) não mostram resultados significativos. No entanto, a variação do MTTR destes dois componentes mostram um pequeno mas perceptível impacto sobre a disponibilidade do sistema (ver Figuras 6.1(d) e 6.1(f), respectivamente).

Uma análise de sensibilidade também foi realizada para os mesmos três componentes



(a) Tempo de descarga da bateria

(b) Tempo de recarga da bateria

(c) MTTF GA

(d) MTTR GA

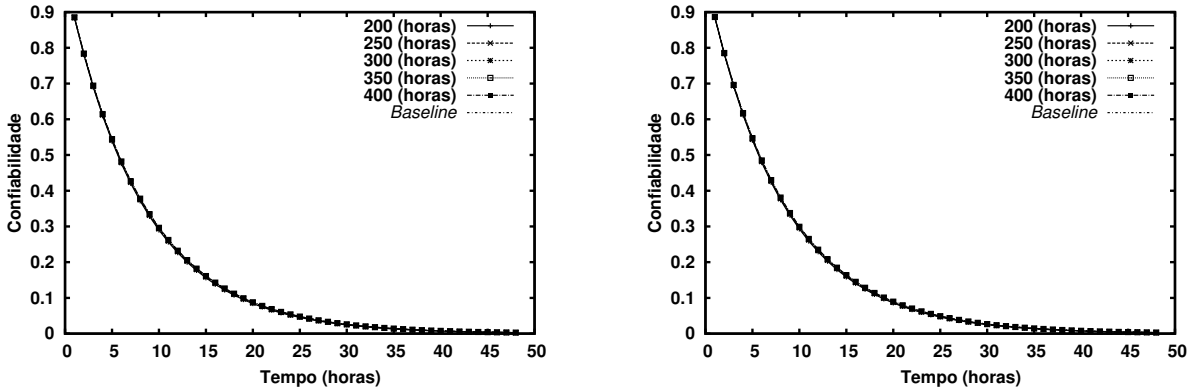
(e) MTTF GI

(f) MTTR GI

Figura 6.1: Disponibilidade para diferentes parâmetros dos principais componentes da MCC

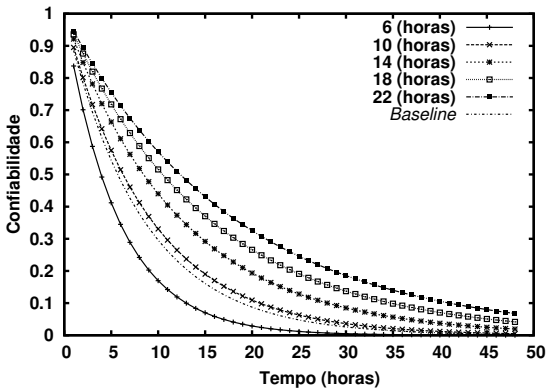
mais importantes considerando a confiabilidade do sistema em um período entre uma e quarenta e oito horas (ou seja, dois dias). Figura 6.2 mostra que a bateria também tem um maior impacto na confiabilidade do sistema do que os outros dois componentes. Como esperado, a confiabilidade diminui à medida que o tempo passa.

As Figuras 6.2(a) e 6.2(b) mostram uma variação paramétrica nos MTTFs dos componentes GI e GA, adotando os seguintes valores: 200, 250, 300, 350, e 400 horas, e também



(a) MTTF GI

(b) MTTF GA



(c) Tempo de descarga da bateria

Figura 6.2: Confiabilidade para diferentes valores de MTTFs dos principais componentes da MCC

inclui o valor de referência (*baseline*) da confiabilidade. Os resultados não mostraram uma mudança significativa em relação ao cenário de referência (*baseline*). Em ambos os gráficos, as seis linhas diferentes da confiabilidade são sobrepostas. Todas eles têm uma confiabilidade de cerca de 88,5% em 1 hora, e apenas 0,02% às 48 horas. No entanto, ao avaliar diferentes tempos de descarga da bateria, é possível observar uma diferença notável em termos de confiabilidade para cada valor possível de tempo de descarga. A Figura 6.2(c) mostra o comportamento da confiabilidade através da adoção de valores diferentes para a descarga da bateria: 6, 10, 14, 18 e 22 horas. Há uma considerável diferença na confiabilidade expressada em cada uma das seis curvas (incluindo a confiabilidade do cenário *baseline*), especialmente durante a primeira hora. Para o tempo de descarga de 6 horas, a confiabilidade do sistema já havia caído para 83,74%. Quanto ao tempo de descarga de 10 horas, a confiabilidade foi melhor na primeira hora, com 89,52%. Como esperado, na medida em que o tempo de descarga da bateria aumenta, a confiabilidade também aumenta. No entanto, essa diferença diminui a partir de um intervalo de descarga para o outro (14h: 92,11%; 18h: 93,58%; 22h: 94,53%). Considerando-se a observação de 48 horas, temos a seguinte confiabilidade para cada tempo de descarga: 0,02% para 6 horas; 0,49% para 10 horas; 1,94% para 14 horas; 4,15% para 18 horas; e 6,74 % para 22 horas.

Portanto, é possível concluir que a adoção de baterias com maior capacidade traz uma considerável melhoria na confiabilidade de toda a MCC, especialmente quando comparado com os outros componentes. É importante salientar que o nosso modelo considera a disponibilidade do ponto de vista de um único usuário, por isso é razoável ter a descarga da bateria como um fator importante para o sistema neste contexto. Mesmo que o modelo for adaptado para representar um sistema com muitos usuários, a taxa de descarga da bateria poderia ser um parâmetro único compartilhado entre todos os dispositivos de usuários. Esta suposição é baseada no fato de que os componentes de *hardware* e *software* são geralmente uniformes em ambientes corporativos que dependem de computação móvel. Portanto, o tempo de descarga da bateria ainda poderia ser importante para a disponibilidade de todo o sistema.

Os elevados níveis de disponibilidade e confiabilidade devem ser destinados a garantir a satisfação do cliente e, posteriormente, rendimentos mais elevados. Uma vez que é possível aplicar diferentes estratégias de redundância na nuvem, decidimos realizar uma análise separada para o lado do provedor de nuvem.

Cenários com componentes redundantes:

O objetivo desta análise é avaliar arquiteturas distintas que o provedor de nuvem pode empregar para permitir interrupções mínimas no serviço móvel. Esta avaliação compara cenários distintos com redundância em diferentes componentes da nuvem, e da adoção de estratégias de redundância distintas.

Essa análise compreende mais quatro cenários: o cenário #2A emprega redundância somente para o servidor GI; o cenário #2B emprega redundância somente para o servidor GA; o cenário #2C emprega redundância em ambos os componentes, servidores GI e GA; e o cenário #2D emprega redundância em todos os três principais componentes (bateria, GI, e GA). As estratégias de replicação TMR, *hot-standby*, *warm-standby*, e *cold-standby* foram consideradas para todos os cenários com redundância. A Figura 6.3 apresenta a disponibilidade calculada para cada cenário.

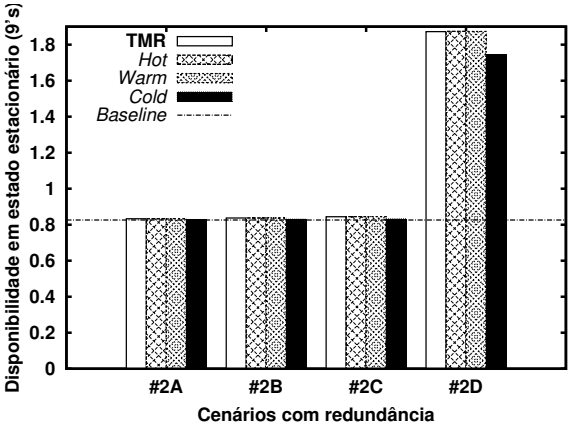


Figura 6.3: Disponibilidade para cenários com redundância

A disponibilidade em estado estacionário aumenta significativamente a partir do cenário *baseline* para o cenário #2D, mostrando a eficácia da redundância. Tal aumento é também verificado através do número de noves, que dá uma perspectiva logarítmica da disponibilidade. O cenário #2A (redundância no componente GI) é muito similar para as estratégias TMR, *hot-standby* e *warm-standby*, pois possuem 0,83319 nove para o TMR, 0,83323 nove para o *hot-standby*, e 0,83319 nove para *warm-standby*. No entanto, possui um valor menor para redundância *cold-standby* (0,82845 nove), que é um valor próximo do cenário base (0,82622 nove). O cenário #2B (redundância no componente GA) possui resultados ligeiramente melhores do que no cenário #2A, com 0,83731 nove (TMR), 0,83741 nove (*hot-standby*), 0,83737 nove (*warm-standby*) e 0,83022 nove (*cold-standby*). Como esperado, o cenário #2C mostraram melhores resultados do que os dois cenários anteriores. Em comparação com o cenário *baseline* (sem redundância), #2C aumentou cerca de 0,02 nove para TMR, *hot-standby*, e *warm-standby*, e cerca de 0,01 nove para redundância *cold-standby*. No entanto, o cenário #2D (redundância na bateria, GI, e GA) alcançou cerca de 1,05 nove de disponibilidade a mais do que o cenário de referência, considerando as estratégias TMR, *hot-standby*, e *warm-standby*, e 0,92 nove a mais do que o *baseline*, considerando a redundância *cold-standby*. Isto indica que a disponibilidade foi aumentada devido à utilização de uma bateria reserva. Os resultados mostram que a redundância apenas no servidor GA (cenário #2B) tem maior impacto sobre a disponibilidade do sistema do que a redundância apenas no servidor de GI (cenário #2A).

Dessa maneira, se um administrador de sistema precisa escolher entre redundância no GI ou GA, a redundância no GA deve ter a máxima prioridade assumindo que a disponibilidade do sistema é o único fator de decisão.

O MTTF para todos os cenários habilitados para a redundância é a mesma do cenário não-redundante. Portanto, a confiabilidade do sistema não foi afetada, apesar do elevado aumento na disponibilidade. Isto ocorre porque o sistema falha na ocorrência de uma única falha no servidor principal, de modo que os cenários redundantes não evitam qualquer falha. Os cenários com servidores redundantes dependem de uma transição rápida para o servidor secundário (ou seja, uma reparação rápida do sistema) para alcançar uma disponibilidade alta.

Análise de custo:

A fim de comparar os custos de uma nuvem privada *versus* uma nuvem pública, consideramos um plano padrão para um aplicativo de mídia usando *Amazon Web Services*, que compreende 32 GB de armazenamento, 2 vCPU, 7.5 de RAM, 10 GB por mês de transferência de dados para dentro da nuvem, e 2 GB por mês de transferência de dados para fora da nuvem. Para mais detalhes sobre as especificações, consulte (AMAZON, 2017). Tabela 6.6 mostra o custo mensal dos recursos necessários.

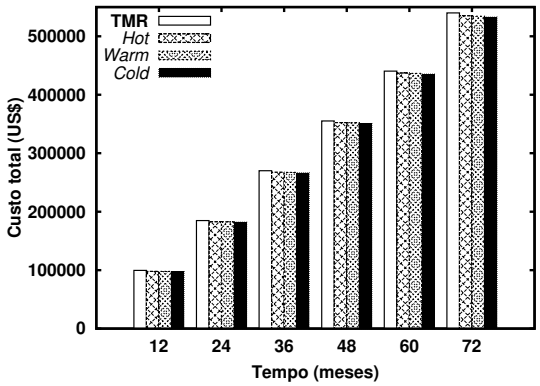
Para calcular o custo de aquisição de equipamentos (σ_s) da infraestrutura de nuvem privada, consideramos um servidor torre DELL/Power Edge T320, com um processador Intel

Tabela 6.6: Estimativa da fatura mensal para a AWS

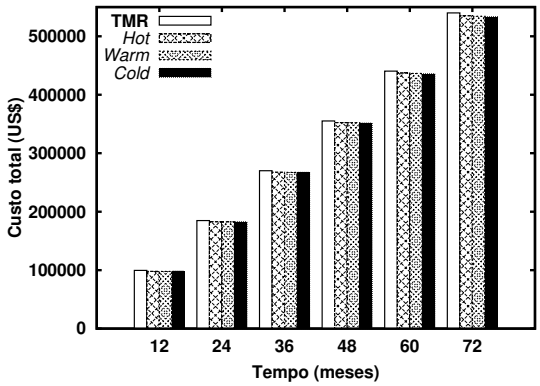
Serviço	**Custo (US$)**
Serviço Amazon EC2 (US-Leste)	530,74
Serviço Amazon S3 (US-Leste)	9,01
Serviço Amazon *CloudFront*	754,96
Transferência de dados para a AWS	0,00
Transferência de dados da AWS	0,27
Suporte AWS (*Business*)	129,50
Pagamento mensal total	1.424,48

Xeon E5-2430L 2 GHz, 16 GB de RAM e 500 GB de disco rígido, que custa US$ 1.244,27 DELL (2016b). Então, todos computadores que compõem a infraestrutura de nuvem possuem a mesma configuração de *hardware*, sendo que a quantidade varia de acordo com o cenário.

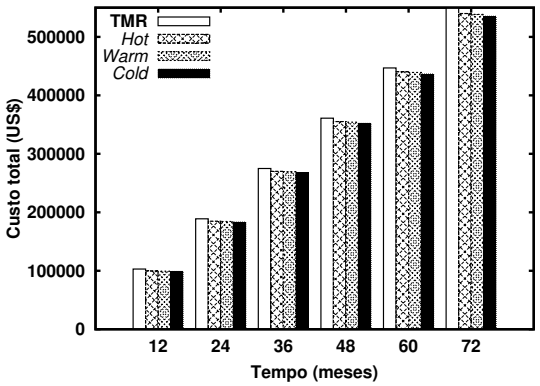
A Figura 6.4 mostra o custo total para os primeiros seis anos dos três cenários de nuvem estudados usando quatro tipos diferentes de redundância adotadas. As despesas de instalação e operação da *cloud* custam cerca de 100 mil dólares no primeiro ano, e pouco mais de 530 mil dólares em seis anos.



(a) Cenário #2A



(b) Cenário #2B



(c) Cenário #2C

Figura 6.4: Custo total para cada cenário

Como a Figura 6.4 está considerando o custo total de toda a infraestrutura de nuvem

para os primeiros seis anos, nós não podemos ver claramente a diferença entre os cenários e as diferentes estratégias de redundância adotadas. Por isso, estamos incluindo separadamente os gráficos com os custos de energia e de reparos da nuvem, que são dois dos custos influenciados principalmente pelo tipo de infraestrutura adotada.

A Figura 6.5 mostra os custos de consumo de energia por ano para cada um dos três cenários: #2A, #2B, e #2C. Os resultados dos cenários #2A e #2B são muito semelhantes para os quatro tipos de redundância adotadas, custando cerca de US$ 1.679,00 para o TMR, US$ 1.493,00 para o *hot-standby*, US$ 1.400,00 para o *warm-standby*, e US$ 1.306,00 para o *cold-standby*, tendo o cenário #2C apresentado um gasto superior que os dois cenários anteriores. O TMR aumentou US$ 373,00, o *hot-standby* aumentou cerca de US$ 186,00, o *warm-standby* aumentou US$ 93,00, e o *cold-standby* aumentou em apenas alguns centavos.

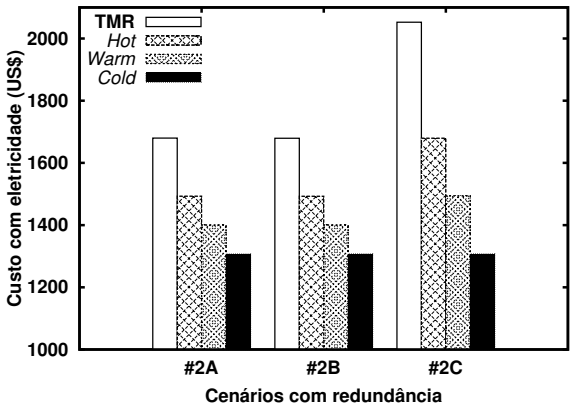


Figura 6.5: Custo anual do consumo de energia elétrica

A Figura 6.6 mostra os custos dos reparos para cada cenário. As despesas com reparo também são as mesmas para os cenários #2A e #2B, e o cenário #2C apresenta novamente um custo mais elevado do que os outros dois cenários. A redundância *cold-standby* tem os mesmos valores para os três cenários, cerca de US$ 980,00. *Hot-standby* e *warm-standby* têm valores idênticos para os cenários #2A e #2B, e também têm valores iguais no cenário #2C (US$ 1.120,00), exceto que neste último cenário ouve um aumento de 140 dólares. O custo do TMR também cresceu no cenário #2C, de US$ 1.260,00 para US$ 1.540,00.

A Figura 6.7 mostra uma comparação da nuvem privada com uma nuvem pública. Os cenários #2A e #2B novamente mostram resultados semelhantes. A nuvem privada é mais viável que a nuvem pública de 48 a 60 meses, tendo o seu primeiro ponto de inflexão entre 36 e 48 meses. Nesses cenários, os custos da infraestrutura de nuvem privada são mais elevados do que alugar uma capacidade semelhante na Amazon até os 36 primeiros meses, aproximadamente. Mesmo com os gastos com a compra de novas máquinas no sexto ano, as estratégias de redundância *warm-standby* e *cold-standby* são ainda mais baratas do que uma nuvem pública. Mas a nuvem privada volta a ser pior do que a nuvem pública em torno de 72 meses, mostrando que há um novo ponto de inflexão entre 60 e 72 meses, devido à substituição de máquinas da nuvem privada. Portanto, a implantação e manutenção de uma infraestrutura de nuvem para um serviço móvel

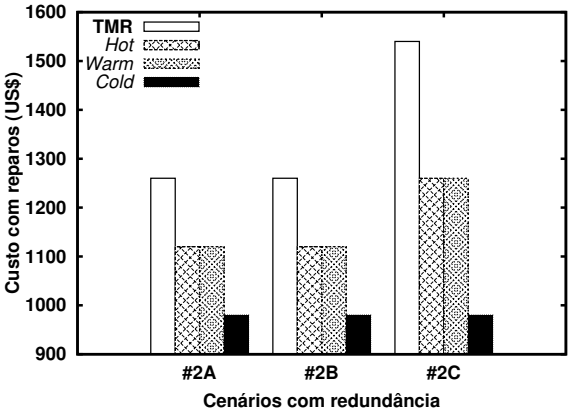
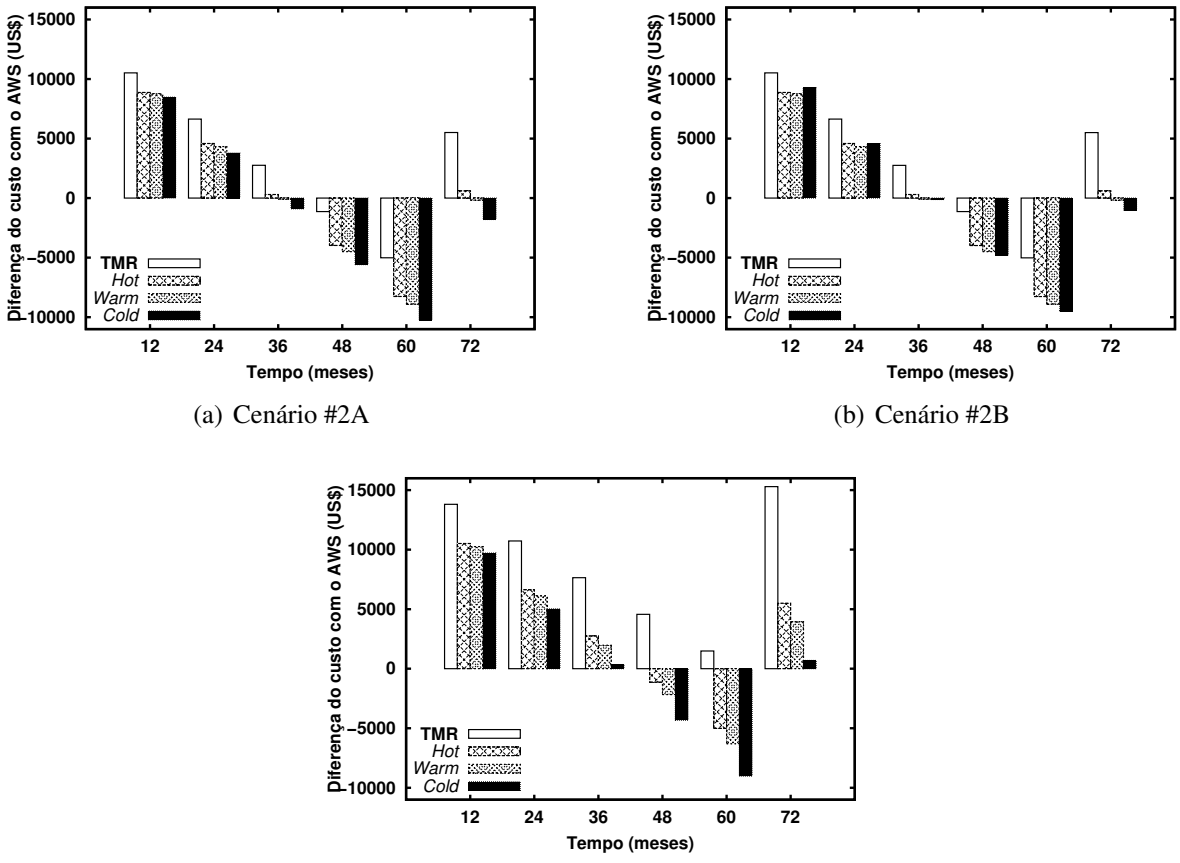


Figura 6.6: Custo anual com reparos

compensa em médio prazo, isto é, para serviços destinados a estar operacional por mais de 36 meses.



(a) Cenário #2A

(b) Cenário #2B

(c) Cenário #2C

Figura 6.7: Diferença dos custos entre cada cenário da nuvem privada *vs* nuvem pública

Para o cenário #2C, os custos de uma nuvem pública só passou a ser maior do que a nuvem privada a partir dos 48 meses para os modelos de redundância *hot-standby*, *warm-standby* e *cold-standby*. Mas, para o TMR não compensa usar uma nuvem privada em qualquer momento. Observamos também que o custo de substituição de computadores no sexto ano faz com que a

despesa acumulada de nuvem privada seja maior do que a nuvem pública.

Os resultados indicam que o custo de nuvens públicas é mais atraente para os serviços que se destinam a trabalhar por um curto período de tempo, ou que pode exigir aumento de capacidade num curto período de tempo.

6.2 Avaliação de disponibilidade e confiabilidade considerando envelhecimento e rejuvenescimento de *software*

Esta seção está dividida em duas partes: resultados experimentais e resultados dos modelos. Os resultados experimentais visam avaliar questões de envelhecimento e rejuvenescimento de *software* considerando a perspectiva do dispositivo móvel e da infraestrutura de nuvem. Já os resultados dos modelos utiliza os resultados experimentais para avaliar todo o sistema de *mobile cloud computing*. Portanto, os resultados dos modelos dependem dos resultados obtidos através de experimentos.

6.2.1 Resultados experimentais

Esta subseção apresenta o resultados experimentais do dispositivo móvel e de uma infraestrutura de nuvem privada. Os experimentos foram planejados e executados obdecendo a metodologia descrita no Apêndice A. Objetivo desses experimentos é investigar a existência do envelhecimento de *software* nesses ambientes, bem como identificar o tempo médio de falha relacionada ao envelhecimento. Os tempos obtidos nesta etapa serão usados como parâmetros de entrada para os modelos de todo o sistema de *mobile cloud computing* propostos mais adiante.

Dispositivo móvel:

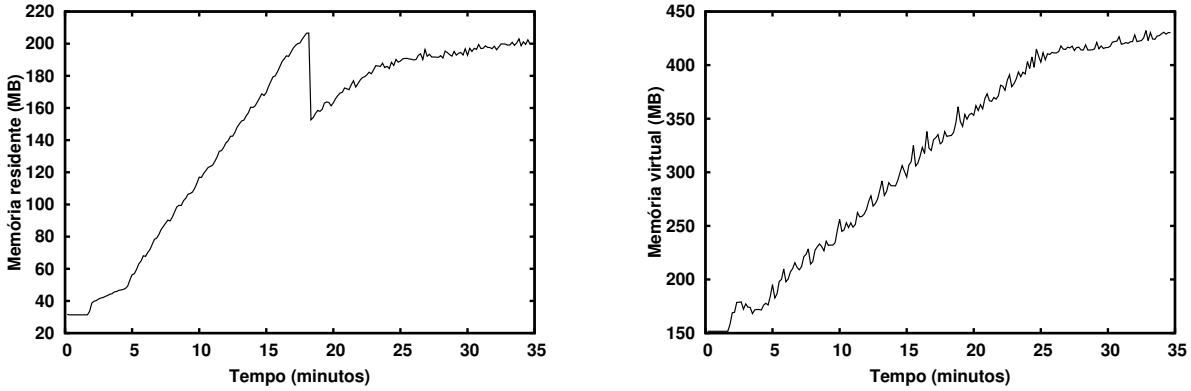
A fim de testar a eficácia da abordagem de investigação de envelhecimento de *software* para dispositivos móveis proposta, um ambiente de testes foi criado em conformidade com as especificações do Apêndice A. O aplicativo Foursquare (FOURSQUARE, 2015) foi selecionado para o experimento por se tratar de uma rede social baseada em localização para dispositivos móveis que está sendo usada por milhões de pessoas em todo o mundo, com troca de mensagens pela rede através de dispositivos móveis e a localização do usuário.

O ambiente de testes utilizou uma máquina *desktop* (Intel Core 2 Duo, CPU E7500 @ 2.93GHz, 4 GB de RAM e um disco rígido de 500 GB SATA) para gerar a carga de trabalho usando o sistema operacional Windows 7 Professional de 64 bits (6,1, Build 7601) Service Pack 1, e um *smartphone* Samsung Galaxy Ace S5830 (800 MHz ARM 11 CPU, 278 MB de RAM, 158 MB de armazenamento interno, e 2GB microSD) para executar o sistema operacional Android v2.3.4 (versão do kernel 2.6.35.7).

Os dados coletados nos experimentos indicam efeitos de envelhecimento significativos no Foursquare. Outros processos importantes para o sistema operacional Android como

system_server e *zygote* também foram considerados, mas como não foram detectados resultados significativos relacionados ao envelhecimento de *software*, tais resultados não foram incluídos na análise. A eficácia do método é demonstrada pelos resultados discutidos abaixo.

A Figura 6.8 mostra a utilização de memória residente e de memória virtual pelo Foursquare (*com.joelapenna.foursquared*) onde grandes aumentos na utilização de ambos os recursos são evidentes. Comparando os respectivos valores no início do experimento e no final, a memória residente teve um aumento de cerca de 200 MB, e a memória virtual aumentou cerca de 425 MB. Tais resultados indicam que, ao aplicar a ferramenta Monkey para imitar as operações de toque do usuário, o processo do Foursquare apresentou efeitos de envelhecimento *software* durante a realização do experimento. Além disso, vale ressaltar que o experimento funcionou por apenas 35 minutos, momento em que o aplicativo parou de funcionar. Portanto, em ambientes reais onde os aplicativos rodam ininterruptamente por horas ou mesmo dias e onde há um maior número de processos diferentes envolvidos, a quantidade de memória desperdiçada seria provavelmente muito maior do que a relatada aqui.



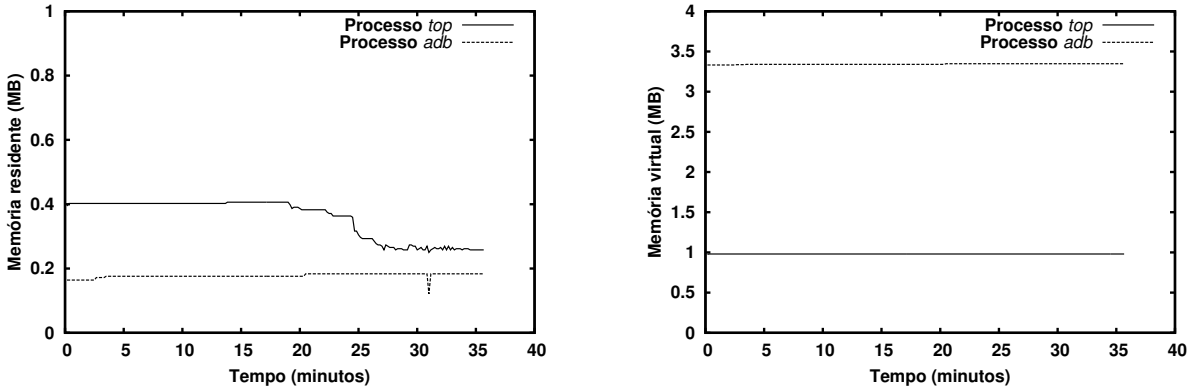
(a) Memória residente (b) Memória virtual

Figura 6.8: Utilização de memória do processo Foursquare

Em SOs *desktop*, a memória virtual basicamente utiliza o disco rígido como uma extensão da memória RAM, mas em dispositivos móveis o sistema operacional Android emprega a memória flash para executar esta tarefa. Além de ser um recurso limitado, a memória *flash* é mais lenta que a memória RAM e, portanto, o tempo de execução da aplicação aumenta à medida que a utilização da memória virtual cresce. Assim, pode-se mostrar que o envelhecimento de *software* não afeta somente a disponibilidade da aplicação, mas também o seu desempenho

Para determinar se a abordagem interferiu no funcionamento do sistema operacional Android, o uso de recursos dos processos *adb* e *top* também foram monitorados. A Figura 6.9 ilustra a utilização das memórias residente e virtual de ambos os processos. Os resultados mostram que a utilização da memória residente é muito pequena. No início do monitoramento, o processo *top* utiliza apenas 0,4 MB, que, em seguida, diminuiu para cerca de 0,3 MB. O processo *adb* começou usando apenas cerca de 0,2 MB, e assim se mantive até o final do experimento com pouca variação. Já a utilização da memória virtual foi um pouco maior do que a memória

residente, mas ainda relativamente baixa. O *top* utilizou cerca de 1 MB e o *adb* utilizou cerca de 3,4 MB. Considerando estes números, pode-se afirmar que a interferência causada pelos mecanismos de monitoramento e de *workload* não é significativa quando comparada com o ambiente como um todo. Além disso, não foram observados sintomas de envelhecimento.



(a) Memória residente (b) Memória virtual

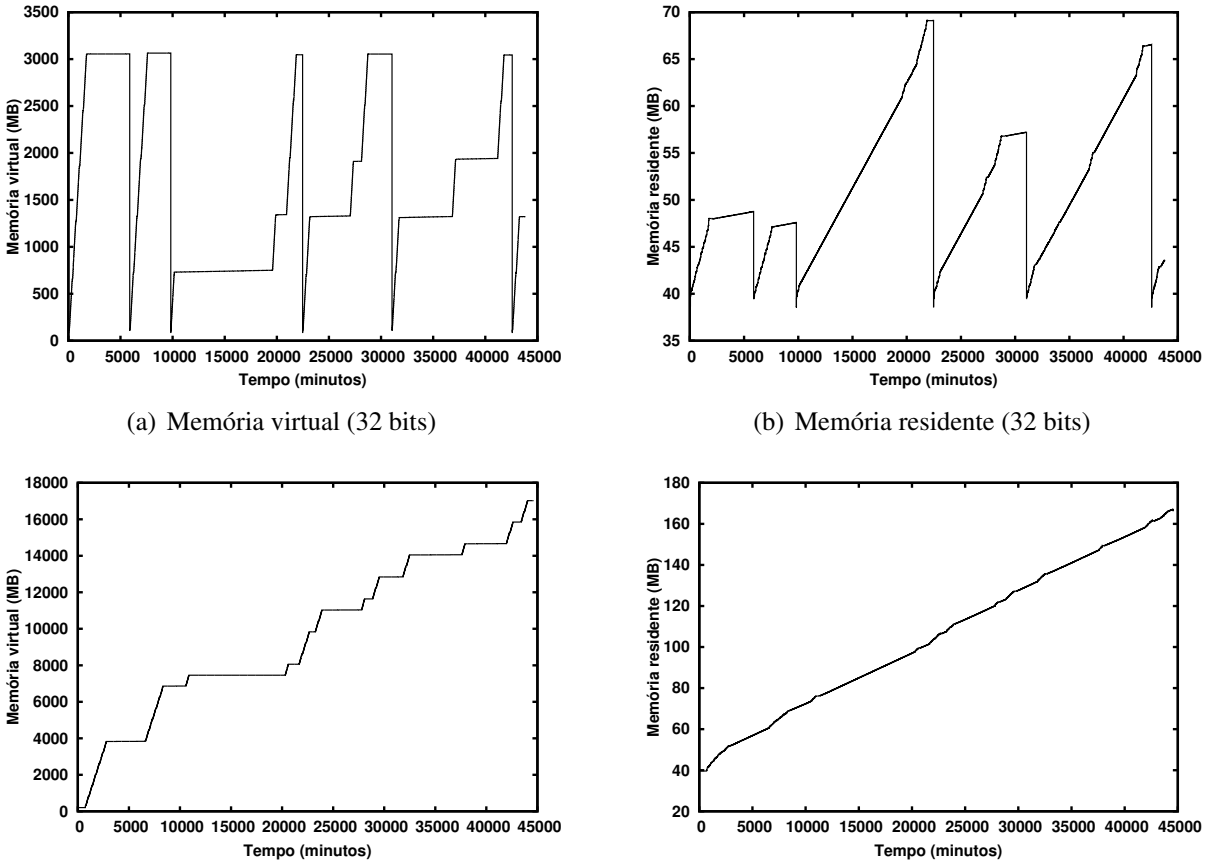
Figura 6.9: Utilização de memória dos processos *top* e *adb*

Computação em nuvem:

Esta subseção tem por objetivo apresentar os resultados obtidos a partir dos experimentos de envelhecimento de *software* e das estratégias de rejuvenescimento descritas no Apêndice A. Para que os recursos fossem monitorados automaticamente e a execução de ações de envelhecimento ocorressem de maneira adequada, foi necessário desenvolver a ferramenta DR Monitor (descrita em detalhes no Apêndice C). Além disso, para automatizar a predição de utilização de recursos através de séries temporais, foi desenvolvida a ferramenta Predictor (descrita em detalhes no Apêndice D), que será utilizada para apoiar a execução das ações de rejuvenescimento de *software*.

Os experimentos realizados evidenciaram um uso cada vez maior de memória virtual e memória residente para o processo do controlador de nó da plataforma de computação em nuvem Eucalyptus. A Figura 6.10 mostra a utilização das memórias virtual e residente de duas máquinas com sistemas operacionais de arquiteturas diferentes, uma com 32 bits e a outra com 64 bits.

O uso das memórias virtual e residente pelo processo de controlador do nó foi aumentando durante a execução dos experimentos. Na máquina de 32 bits, o processo CN chegou a atingir cerca de 3055 MB de memória virtual (ver Figura 6.10(a)), momento em que o processo parou de crescer. Neste ponto, o controlador do nó não pôde mais instanciar novas máquinas virtuais, provavelmente devido às limitações de utilização de memória virtual do sistema operacional de 32-bits. Com isso, foi necessário executar uma reinicialização manual do processo controlador do nó, o que ocasionou a queda do uso de memória para menos de 110 MB. Depois que o



(a) Memória virtual (32 bits)

(b) Memória residente (32 bits)

(c) Memória virtual (64 bits)

(d) Memória residente (64 bits)

Figura 6.10: Utilização de memória do processo *Node Controller* do Eucalyptus

processo foi reiniciado, o mesmo padrão se repetiu: a memória virtual do processo cresceu novamente até cerca de 3064 MB e novamente o controlador de nó não foi capaz de atender os pedidos de instanciação de novas máquinas virtuais. Outro recurso monitorado nessa máquina foi o uso de memória residente no controlador do nó. De acordo com a Figura 6.10(b), podemos notar semelhanças na utilização de memória residente com a memória virtual. O crescimento do uso da memória residente também é evidente, e as quedas de utilização de memória também estão relacionadas com o momento em que o controlador do processo é reiniciado. No entanto, esse crescimento não chegou a tomar maiores proporções, pois o processo falhou antes mesmo de chegar a um nível crítico por conta do crescimento de memória virtual.

Na máquina com arquitetura de 64 bits, também observamos uma utilização crescente de memória virtual, como mostrado na Figura 6.10(c). Neste caso, não há nenhuma queda na utilização dos recursos, pois o processo do controlador do nó não chegou a falhar, de modo que o processo não foi reiniciado. Tal fato confirma que o problema notado na máquina de 32 bits foi causado por limitações da arquitetura. Não há evidência que demonstra quando esse sistema poderia atingir um nível crítico, ou seja, o ponto onde ele vai falhar, pois sabe-se que a arquitetura de 64 bits permite a utilização de até 256 TB de memória virtual. No entanto, se o uso de memória virtual tivesse continuado a crescer ao longo de um período mais longo, o espaço

no disco rígido usado pelo sistema de gerenciamento de memória do Linux estaria esgotado, além da perda de desempenho ocasionada por operações de leitura e escrita em disco. Já o uso da memória residente (ver Figura 6.10(c)) obteve um crescimento quase linear. Também não observamos qualquer queda na utilização de memória, porque, como mencionado anteriormente, esta máquina não chegou a falhar. Se o uso da memória residente continuasse a crescer por um período mais longo, provavelmente, a memória RAM usada pelo sistema de gerenciamento de memória do Linux seria esgotada.

Após identificar o envelhecimento de *software* na plataforma Eucalyptus, precisamos verificar a eficácia da estratégia de rejuvenescimento proposta. Os dados coletados no experimento da Figura 6.10 foram utilizados para descobrir qual o tipo de série temporal tem o melhor ajuste para o crescimento do uso de memória virtual em processos do controlador do nó. Iremos focar no rejuvenescimento do processo do controlador do nó, uma vez que mostrou os principais efeitos do envelhecimento entre todos os componentes monitorados.

Foram utilizados quatro modelos (LTM, QTM, GCM e SCTM) para análise de tendências. Um resumo dos resultados da série e suas taxas de erro são apresentados na Tabela 6.7, onde $\widehat{Y}_t$ é o valor previsto do consumo de memória no tempo de t.

Tabela 6.7: Resumo dos índices de precisão para cada modelo

Modelo	$\widehat{Y}_t$	**MAPE**	**MAD**	**MSD**
LTM	$44157.1 + 2.85t$	1%	900	1472447
QTM	$43354.8 + 2.95830t - 0.000002t^2$	1%	872	1343698
GCM	$53013.6(1.00003^t)$	6%	6014	52619294
SCTM	$(10^6)/(4.70218 + 16.8036(0.999942^t))$	2%	1259	3449812

Podemos observar na Tabela 6.7 que os valores dos índices MAPE, MAD e MSD são menores para os modelos LTM e QTM. Assim, a escolha deve ser feita em relação a esses dois modelos. Observa-se também que, apesar dos valores dos índices MAPE serem os mesmos para estes dois modelos, os valores de índice do modelo QTM são menores do que para os do modelo LTM. Assim, o modelo QTM foi escolhido como o melhor ajuste para a análise de tendências de utilização da memória virtual no controlador do nó do Eucalyptus, e, portanto, ele foi usado para agendar o rejuvenescimento.

O estudo experimental para a verificação do método de rejuvenescimento foi realizado em duas partes. Na primeira, o ambiente de nuvem recebeu uma carga de trabalho com o objetivo de exaustar os recursos computacionais, e o mecanismo de rejuvenescimento só foi acionado quando o *script* de monitoramento detectou que o limite crítico foi alcançado. No segundo experimento, foi utilizada a mesma carga de trabalho, mas o rejuvenescimento foi programado com base nas previsões de séries temporais. Maiores detalhes sobre os experimentos estão presentes no Apêndice A.2)

A Figura 6.11 mostra a análise de tendência para o crescimento da utilização da memória virtual utilizando a função quadrática $\widehat{Y}_t = 94429 + 3825.3t - 0.0686t^2$. Esta análise identificou

o valor de 809 minutos para a Utilização de Memória Crítica (UMC) T_{UMC}, isto é, o tempo previsto para atingir o limite de 3 GB. Com isso, uma ação de rejuvenescimento foi agendada para $T_{rej} = 809 - (5/60 + 5)$, em minutos, contados a partir do início do experimento. Depois do rejuvenescimento, o uso de memória é reduzido, e outra análise de tendências é realizada quando o limite de 80% for atingido novamente.

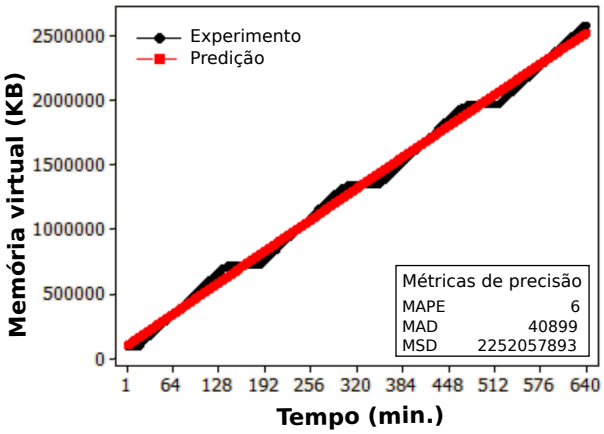


Figura 6.11: Análise de tendência da memória virtual

Os resultados mostram que a estratégia de rejuvenescimento proposta traz a disponibilidade do sistema para um nível mais elevado, quando comparado com o método que não se baseia na análise de tendências com séries temporais. O número de noves aumentou de 3,38 para 4,11 (ver Tabela 6.8). Em um lapso de tempo de um ano, essa diferença significa uma diminuição de cerca 218 minutos para 40 minutos de tempo de inatividade, ou seja, o tempo de indisponibilidade foi reduzido em cerca de 80%, evitando que os pedidos de novas instâncias de máquinas virtuais sejam rejeitadas, mesmo na existência de recursos computacionais suficientes para instancia-las.

Tabela 6.8: Comparação dos experimentos

Métrica	**Experimento #1**	**Experimento #2**
Disponibilidade (%)	99,9584	99,9922
Disponibilidade (9's)	3,38	4,11
Downtime (segundos)	108	20

É interessante destacar que a utilização de séries temporais para reduzir o *downtime* do sistema durante a realização de uma ação de rejuvenescimento pode também ser aplicada a outros ambientes de computação. A série temporal em si não está envolvida diretamente no sistema, mas indica o momento em que uma ação deve ser tomada.

6.2.2 Resultados dos modelos

Esta subseção tem por objetivo identificar o impacto do envelhecimento de *software* em uma infraestrutura de *mobile cloud computing*. Para isso, iremos utilizar os modelos propostos

na Subseção 5.2 em conjunto com os resultados obtidos através dos experimentos apresentadas na subseção anterior.

A análise dos modelos da *mobile cloud* considera um cenário de referência (*baseline*), onde não há qualquer falha associada ao envelhecimento de *software* e, portanto, não é necessário aplicar qualquer política de rejuvenescimento. A análise inclui outros quatro cenários considerando envelhecimento de *software* com e sem políticas de rejuvenescimento: no cenário #1, apenas o dispositivo móvel sofre de envelhecimento; no cenário #2, apenas o componente *front-end* da infraestrutura de nuvem sofre envelhecimento; no cenário #3, apenas os nós da infraestrutura de nuvem envelhecem; finalmente, no cenário #4, todos os componentes de *software* sofrem de envelhecimento.

A Tabela 6.9 mostra os parâmetros para os cenários que consideram o envelhecimento e rejuvenescimento de *software*. A taxa de falhas relacionadas com o envelhecimento de *software* para o dispositivo móvel (λ_{mttarf_dm}) no valor de 34,66 minutos obtida anteriormente foi multiplicada por um fator de 100 para obtermos um valor mais realista, já que o estudo mencionado utilizou uma carga de trabalho muito estressante para acelerar o surgimento dos sintomas do envelhecimento. A fim de calcular a probabilidade de sucesso da ativação do agente de rejuvenescimento para o dispositivo móvel (P_{mru_dm}), adotamos as taxas de falhas e reparos de um aplicativo móvel padrão, e usamos a Equação $P_{mru\_dm} = \frac{\lambda_{app}}{\lambda_{app}+\mu_{app}}$. E, a probabilidade do agente não acionar a ação de rejuvenescimento (P_{mrd_dm}) é obtida a partir de $P_{mrd\_dm} = 1 - P_{mru\_dm}$. A taxa de falhas relacionadas com o envelhecimento para o componente da infraestrutura de nuvem (λ_{mttarf_in}) também foi dividida por um fator de 10 para um valor mais realista, uma vez que a carga de trabalho foi adotada para acelerar os sintomas do envelhecimento. Afim de calcular a probabilidade do agente de rejuvenescimento para os componentes de nuvem estar funcionando (P_{mru_in}), adotamos os valores padrão de um componente de nuvem, e usamos a Equação $P_{mru\_in} = \frac{\lambda_{in}}{\lambda_{in}+\mu_{in}}$. E, a probabilidade do agente de rejuvenescimento estar indisponível (P_{mrd_in}), é obtida a partir de $P_{mrd\_in} = 1 - P_{mrd\_in}$). Para evitar que o sistema falhe antes da ação de rejuvenescimento ser acionada, iremos considerar que os tempos de ativação das ações de rejuvenescimento baseadas no tempo (λ_{ar_dm} e λ_{ar_in}), são cerca de 10% menor do que a taxa de falha associada ao envelhecimento. Os demais componentes do dispositivo móvel, comunicação móvel e infraestrutura de nuvem adotaram os parâmetros constantes nas Tabelas 6.1, 6.2 e 6.3.

A Figura 6.12 mostra a disponibilidade para todos os cenários. Como esperado, o cenário #4 apresenta os piores resultados, obtendo uma disponibilidade de apenas 0,8388 (83,88%), quando não há o rejuvenescimento. O cenário #3 foi o menos afetado com o envelhecimento *software*, atingindo 84,88% de disponibilidade, sendo o mais próximo do cenário de referência, que é 84,89%. O Cenário #2 foi o segundo melhor (84,27%), seguido pelo cenário #1 (84.51%). Em todos os cenários, quando o rejuvenescimento de *software* é aplicado, os resultados são muito semelhantes aos do cenário de referência, diferindo apenas na quinta casa decimal. Isto mostra que a política de rejuvenescimento é capaz de trazer a disponibilidade do sistema afetado pelo envelhecimento de *software* para o mesmo nível de um sistema sem efeitos de envelhecimento.

Tabela 6.9: Parâmetros de entrada para os modelos de envelhecimento e rejuvenescimento de *software*

Parâmetro	Valor
λ_{mttarf_dm}	1/57,77 (h^{-1})
μ_{mttarf_dm}	1/0,25 (h^{-1})
λ_{ar_dm}	1/52,00 (h^{-1})
μ_{ar_dm}	1/0,0027778 (h^{-1})
P_{mru_dm}	0,99925805
P_{mrd_dm}	0,00074195
λ_{mttarf_in}	1/134,8333 (h^{-1})
μ_{mttarf_in}	1/1,0 (h^{-1})
λ_{ar_in}	1/121,35 (h^{-1})
μ_{ar_in}	1/0,0027778 (h^{-1})
P_{mru_in}	0,99873257
P_{mru_in}	0,00126743

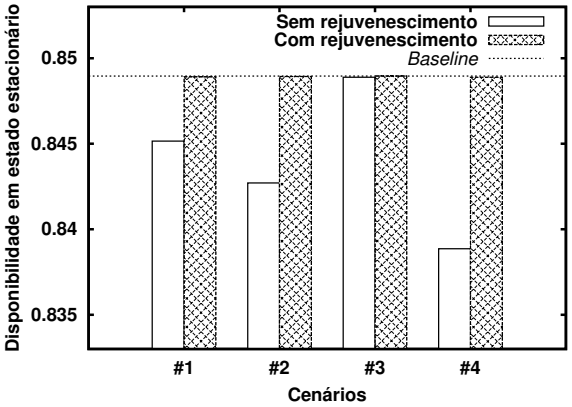


Figura 6.12: Disponibilidade anual da *mobile cloud*

A fim de compreender melhor o impacto do envelhecimento de *software* e de de políticas de rejuvenescimento em todo o sistema (*baseline*), a Figura 6.13 mostra o *downtime* para todos os cenários. O *downtime* do sistema sem envelhecimento de *software* é de 1.324,011 horas. Quando comparado com o cenário *baseline*, o cenário #4 tem cerca de 88,493 horas de tempo de *downtime* anual causado especificamente pelo envelhecimento de *software*, e de cerca de 1.410 horas em tempo de indisponibilidade total. Os cenários #1 e #2 têm 33,338 horas e 54,786 horas de tempo de indisponibilidade causado pelo envelhecimento, respectivamente. No cenário #3, o impacto do envelhecimento sobre a disponibilidade do *software* é mínimo, causando apenas 0,621 horas de tempo de indisponibilidade em um ano, ou seja, o tempo de indisponibilidade total está muito perto do cenário de referência.

No entanto, temos uma grande melhoria nos resultados quando se aplica uma política de rejuvenescimento. Comparando novamente com o cenário *baseline*, o cenário #1 tem apenas 0,390 hora de indisponibilidade, o equivalente a 23,378 minutos a mais do que o tempo de indisponibilidade do cenário (*baseline*). O cenário #2 rendeu 0,221 hora de indisponibilidade

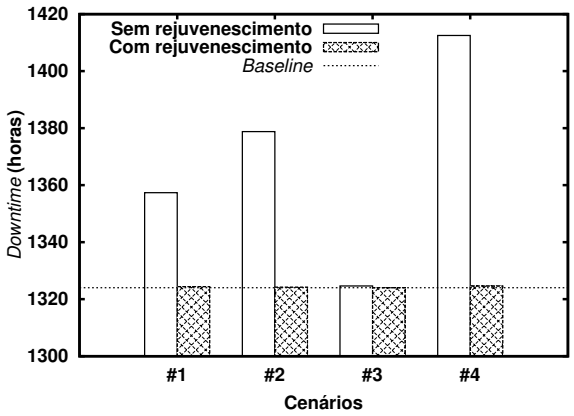


Figura 6.13: *Downtime* anual do sistema

(13,285 minutos). O cenário #3 tem o melhor resultado de todos, com apenas 3,219 segundos de interrupções em um ano inteiro devido ao envelhecimento e rejuvenescimento de *software*. Finalmente, o cenário #4 chegou a uma indisponibilidade de 0,612 horas, ou seja, 36,716 minutos.

A Figura 6.14 mostra a confiabilidade para todos os cenários, mas não considera as políticas de rejuvenescimento, pois o sistema falha na ocorrência de qualquer falha em qualquer componente. Mesmo evitando a falha do componente com uma ação de rejuvenescimento, neste trabalho estamos considerando que a estratégia de rejuvenescimento adotada provoca um tempo de inatividade do sistema, mesmo que por um curto período de tempo. Assim, a confiabilidade do sistema não foi afetada pela ação de rejuvenescimento, apesar do grande aumento na disponibilidade.

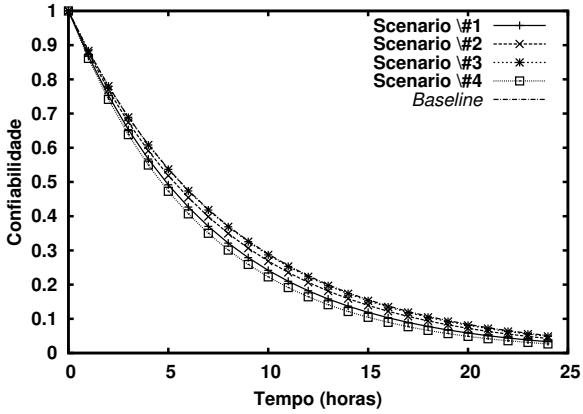


Figura 6.14: Confiabilidade da *mobile cloud*

Todos os cenários tem uma confiabilidade de cerca de 88 em 1 hora, e apenas 3% em 24 horas. No entanto, em 12 horas, observamos uma boa diferença entre os cenários. Neste momento, a confiabilidade do cenário #3 (22,31%) é a que está mais próxima do cenário de *baseline* (22,58%). Como esperado, o cenário #4 tem o pior confiabilidade, com 16,44% em 12 horas de execução do sistema. O cenário #1 é o segundo pior com 18,20%, seguido pelo cenário #2 com 20,66%.

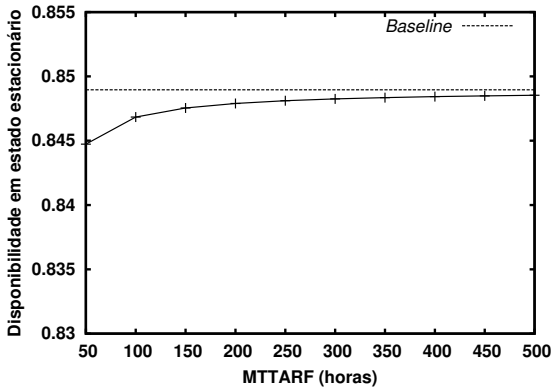
Realizou-se uma análise de sensibilidade para identificar qual componente da MCC é o mais afetado pelas mudanças nas taxas de envelhecimento e rejuvenescimento de *software*. Neste caso, não vamos incluir o cenário #4, uma vez que ele considera a existência simultânea de envelhecimento de *software* em vários componentes. A Figura 6.15 mostra uma variação paramétrica no tempo médio para falhas relacionadas ao envelhecimento de *software* (do inglês, *Mean Time to Aging-related Failure* (MTTARF)) e tempo médio para ações de rejuvenescimento (do inglês, *Mean Time to Rejuvenation Action* (MTTRA)) dos componentes nos três primeiros cenários.

Considerando o MTTARF, o cenário #2 (envelhecimento no *front-end*) foi o mais afetado com a variação do MTTARF (ver Figura 6.15(c)). A disponibilidade começa com 0,8323 (83,23%) com um MTTARF de 50 horas, e vai para 0,8472 (84,72%) com um MTTARF de 500 horas. O cenário #1 (envelhecimento no dispositivo móvel) foi o segundo mais afetado, passando de 84,47% para 84,85%. No entanto, considerando o MTTRA, o cenário #1 é o mais afetado com a alteração no parâmetro de rejuvenescimento. Na Figura 6.15(b), a disponibilidade começa com 0,8482 (84,82%), com um MTTRA de 50 horas, e cai para 0,8414 (84,14%), com uma MTTRA de 500 horas. Figuras 6.15(e) e 6.15(f) mostram que as variações no MTTARF e MTTRA do cenário #3 não tem impacto significativo sobre a disponibilidade de todo o sistema.

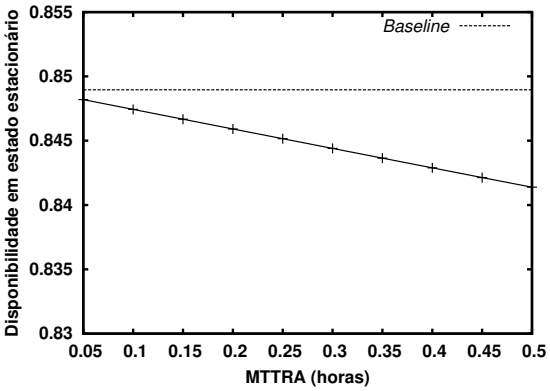
A Figura 6.16 mostra os efeitos sobre a confiabilidade, considerando uma variação paramétrica no MTTARF dos componentes em todos os cenários, adotando os seguintes valores: 100, 200, 300, 400 e 500 horas. A confiabilidade do *baseline* também está incluída nesses gráficos.

Tomando como base o cenário *baseline*, os resultados não mostraram uma mudança significativa em relação ao cenário #3, pois as cinco linhas diferentes de confiabilidade estão sobrepostas. No cenário #1, é possível observar uma pequena diferença apenas quando comparamos o menor valor com os valores maiores. No tempo 12 horas, a linha de MTTARF igual a 100 atinge 20% de confiabilidade, e a linha de valor 500 atinge 22% (ver Figura 6.16(a)). No entanto, nos cenários #2 e #4, pode ser notada uma grande diferença na confiabilidade. No cenário #2 (ver Figura 6.16(b)), a confiabilidade atingiu 17,7% em 12 horas de execução do sistema para um MTTARF igual a 100 horas, e atinge 22% para um MTTARF de 500 horas. No cenário #4 (ver Figura 6.16(d)), novamente em 12 horas, a confiabilidade atinge 17,4% para um MTTARF de 100 horas, e atinge 21,5% para um MTTARF de 500 horas.

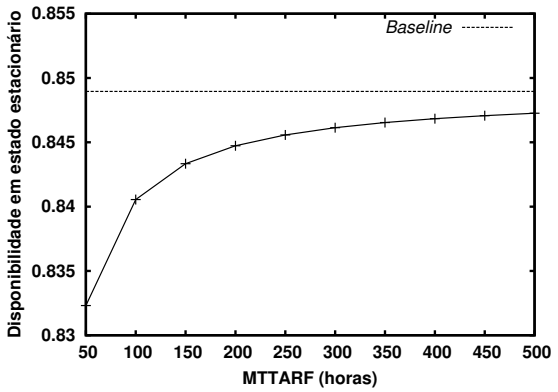
Portanto, podemos notar que a disponibilidade e a confiabilidade do sistema móvel são mais afetadas pelo envelhecimento de *software* quando o *front-end* da nuvem está envolvido (cenários #2 e #4) do que em outros cenários que englobam sintomas de envelhecimento apenas em outros componentes (ou seja, dispositivo móvel e nós da nuvem). Isso destaca a importância de dar enfoque no rejuvenescimento de *software* para o *front-end* nuvem para que esse tipo de falha não tragam consequências danosas para as atividades do sistema.



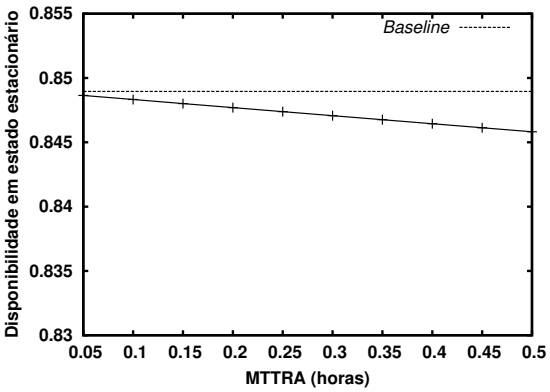
(a) MTTARF cenário #1



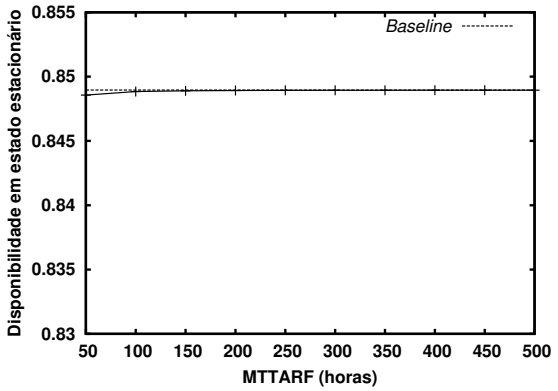
(b) MTTRA cenário #1



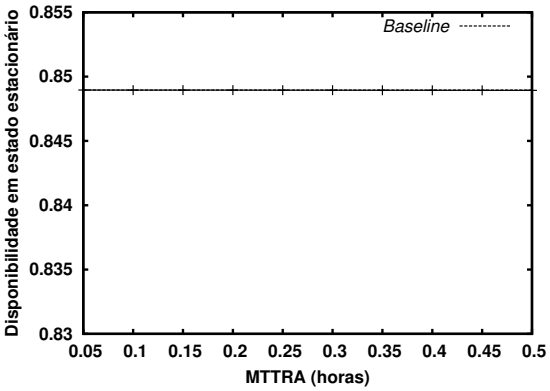
(c) MTTARF cenário #2



(d) MTTRA cenário #2



(e) MTTARF cenário #3

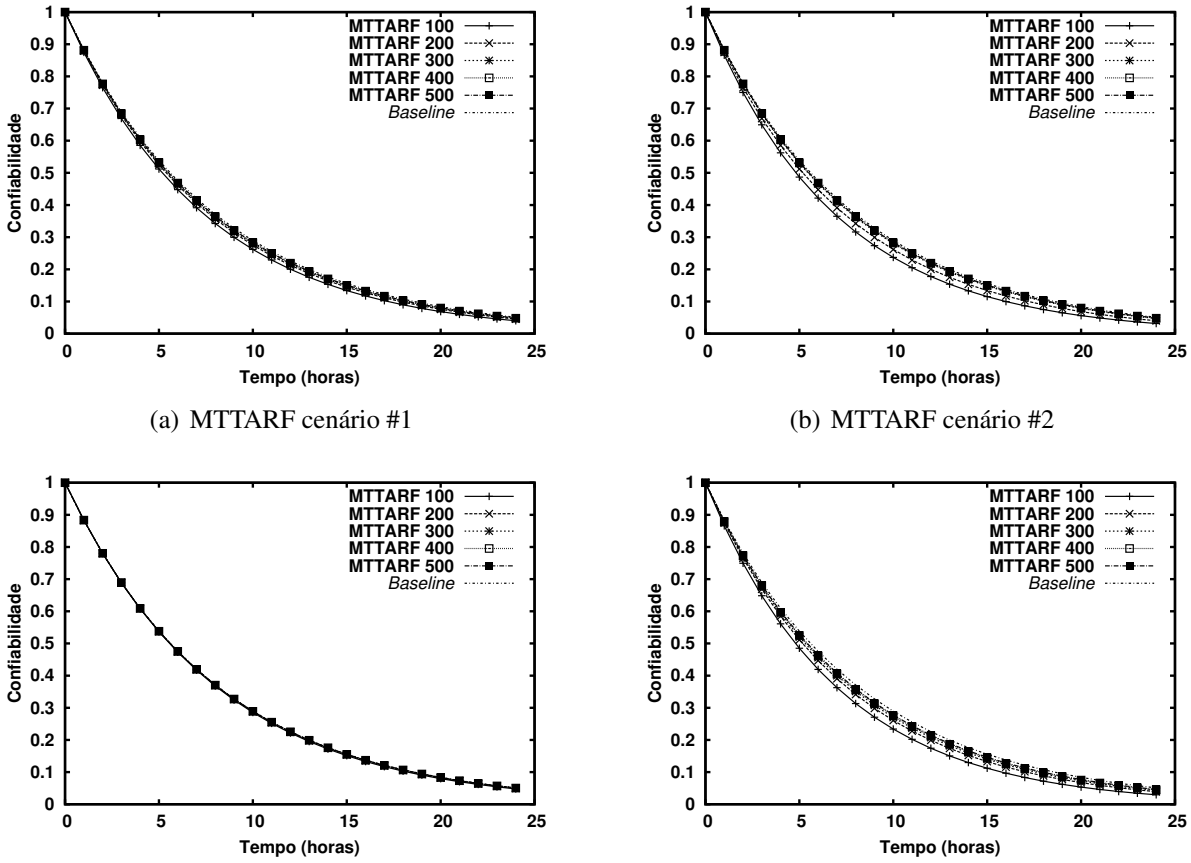


(f) MTTRA cenário #3

Figura 6.15: Variação paramétrica - disponibilidade

6.3 Avaliação do tempo de vida da bateria em dispositivos móveis

Estudos de casos distintos são apresentados nesta seção para avaliar o tempo de vida da bateria em sistemas móveis. A primeira parte desse estudo tem como objetivo identificar o tempo médio de descarga da bateria para a rede local e a rede móvel, considerando os quatro tipos de procolos escolhidos. A segunda parte tem por objetivo avaliar a conectividade das redes de comunicação, considerando cenários com taxas diferentes para áreas de cobertura e



(a) MTTARF cenário #1

(b) MTTARF cenário #2

(c) MTTARF cenário #3

(d) MTTARF cenário #4

Figura 6.16: Variação paramétrica - confiabilidade

não-cobertura. Estes resultados servem como parâmetros de entrada para identificar o tempo médio de um ciclo de vida da bateria e o tempo de vida da bateria. A terceira parte da nossa análise está focada em diferentes taxas para a capacidade da bateria. O principal objetivo é estimar o impacto sobre a vida útil da bateria a partir dos modelos propostos na Seção 5.3.

6.3.1 Parâmetros de entrada

A Tabela 6.1 mostra os parâmetros de entrada para o modelo do dispositivo móvel. A Tabela 6.2 mostra os parâmetros de entrada para componentes comuns para todos os cenários avaliados pelo modelo de conectividade mostrados na Figura 5.11.

A Tabela 6.10 mostra os parâmetros de entrada com valores diferentes para todos os cenários avaliados pelo modelo de conectividade mostrados na Figura 5.11. Os valores adotados para esses parâmetros de entrada (λ_{ln_c}, μ_{ln_c}, λ_{mn_c}, e μ_{mn_c}) tentam representar o comportamento de cada um dos cenários com diferentes características de conectividade; onde o cenário #1 tem boa conectividade para ambas as redes de comunicação, mas utiliza a rede local como conexão preferencial; o cenário #2 tem boa conectividade para a rede local, e conectividade ruim para a rede móvel; o cenário #3 possui boa conectividade para a rede móvel, e conectividade

ruim para a rede local; e, finalmente, o cenário #4 tem conectividade ruim para ambas as redes. A Figura 5.9 ilustra possibilidades reais onde tais diferentes características de conectividade podem ser encontradas.

Tabela 6.10: Parâmetros de entrada para o modelo de conectividade com diferentes cenários

Cenário	**Parâmetros** (h^{-1})			
	λ_{rl_c}	μ_{rl_c}	λ_{rm_c}	μ_{rm_c}
#1	$1/10,0$	$1/0,5$	$1/10,0$	$1/0,5$
#2	$1/10,0$	$1/0,5$	$1/0,5$	$1/10,0$
#3	$1/0,5$	$1/10,0$	$1/10,0$	$1/0,5$
#4	$1/0,5$	$1/10,0$	$1/0,5$	$1/10,0$

Sabemos que os parâmetros de conectividade para diferentes áreas dependem de vários fatores como a distância geográfica do emissor de sinal, da existência de obstáculos físicos e/ou magnéticos, fatores climáticos, mobilidade, receptor, etc. Como não haviam dados de medição disponíveis para os ambientes descritos, foram definidos os parâmetros relatados na Tabela 6.10 com base em sua representatividade.

Vários fatores podem influenciar a vida útil da bateria, como a qualidade das células da bateria, temperatura elevada e altas correntes. Sabe-se também que a capacidade da bateria diminui à medida que o número de ciclos aumenta (BATTERYUNIVERSITY, 2015), e que isto pode influenciar o tempo total de descarga, assim como a recarga completa da bateria. No entanto, para calcular o tempo médio de vida da bateria, adotamos β=1000, na Equação 5.25, como uma estimativa para o número de ciclos da bateria, porque este valor representa um estado em que a bateria ainda está trabalhando em 100-80% da sua capacidade total (BATTERYUNIVERSITY, 2015).

Os tempos médios de descarrega para cada protocolo de comunicação serão obtidos a partir de um estudo experimental (ver Seção 6.3.2) que investigará o consumo de energia dispendida durante o uso contínuo do dispositivo móvel na mesma rede. Os outros parâmetros necessários para o modelo de descarga da bateria serão obtidos através dos resultados do modelo de conectividade (ver Seção 6.3.3).

6.3.2 Resultados experimentais

O ambiente de testes foi construído com apoio do gerador de carga de trabalho e aplicativo cliente móvel para receber mensagens. Foram realizados alguns experimentos de monitoramento em um sistema real. Foram utilizados dois parâmetros variáveis sobre os experimentos: o protocolo de troca de mensagens, e o modo de transferência de dados (redes local e móvel). Cada combinação de valores para esses parâmetros foi definida como um cenário de experimentos. A lista de todos os cenários pode ser observada na Tabela 6.11.

Tabela 6.11: Cenários de consumo de bateria

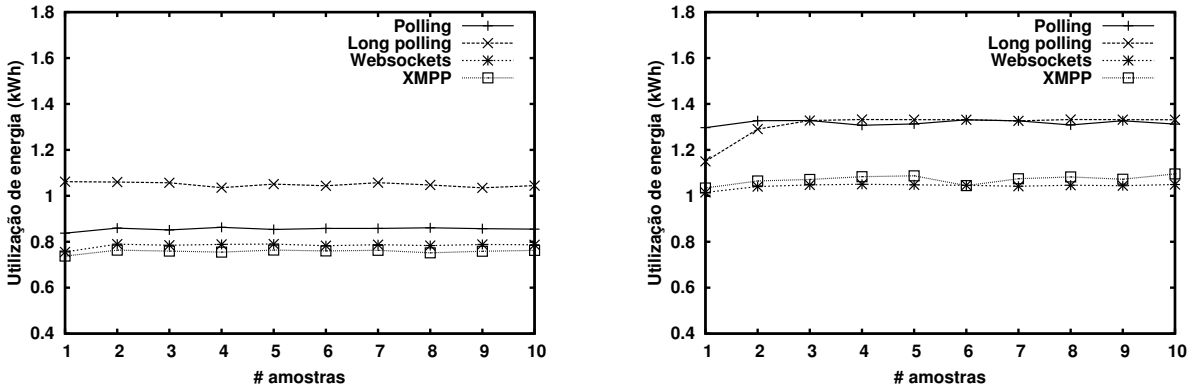
Cenário	Protocolo	Tipo de rede
1	Polling	Local
2	Long polling	Local
3	WebSocket	Local
4	XMPP	Local
5	Polling	Móvel
6	Long polling	Móvel
7	WebSocket	Móvel
8	XMPP	Móvel

Os estudos de caso consistiram nas seguintes etapas. Primeiro, foram realizados experimentos a fim de obter as taxas de consumo de energia para cada protocolo de troca de mensagens associados com diferentes interfaces de redes sem fios. Em seguida, o modelo de descarga da bateria (ver Figura 5.10) foi alimentado a partir dos dados obtidos com os experimentos e dos resultados do modelo de conectividade (ver Figura 5.11). Após, compararam-se os diferentes cenários e foi determinado o protocolo mais eficiente em termos de consumo de energia. Por último, identificamos o tempo de vida da bateria, e avaliamos a disponibilidade do sistema móvel.

Adotamos um prazo de execução de uma hora para cada experimento. Foram coletados um total de 7.200 amostras de consumo de energia instantânea. Com essas amostras, calculamos a energia consumida em cada experimento aplicando a Equação B.3. A carga de trabalho gerada durante os experimentos e o mecanismo de monitoramento adotado foram descritos no Apêndice B.

A Figura 6.17 mostra o consumo de energia dos clientes móveis. Os protocolos apresentaram comportamento distintos em cada experimento. A Figura 6.17(a) mostra o consumo de energia dos clientes móveis usando conexão sem fio numa rede WLAN. O protocolo *XMPP* se mostrou mais eficiente que os demais quando usando uma rede local. Já a Figura 6.17(b) apresenta o consumo de energia para cada protocolo usando a rede móvel. Neste cenário, o dispositivo móvel foi fixado no mesmo ponto geográfico para conservar as mesmas propriedades de conexão. Sem variações em transferências de dados, outras tecnologias como GPRS, EDGE ou 4G, o dispositivo móvel foi associado à mesma estação-base evitando troca de células de comunicação durante o experimento. Qualquer tráfego com serviço de voz foi bloqueado para evitar interferência nos resultados.

Os resultados mostraram que o *Polling* consome mais energia, e que o *WebSocket* foi o mais eficiente. O *Long polling* e o *XMPP* tiveram o consumo de energia intermediária. A Tabela 6.12 mostra um resumo dos tempos de descarga para cada protocolo de comunicação considerado neste trabalho, que servirão como parâmetros de entrada para o modelo mostrado na Figura 5.10.



(a) Rede local (b) Rede móvel

Figura 6.17: Consumo de energia de protocolos de troca de mensagens

Tabela 6.12: Tempos de descarga dos protocolos de comunicação

Protocolo	Rede	Consumo (Wh)	Ciclo de vida (h)	Consumo (W)
Polling	Local	0,85542278	5,19	0,16482134489
	Móvel	1,31776778	3,37	0,39102901483
Long-polling	Local	1,04927000	4,23	0,24805437352
	Móvel	1,30852722	3,39	0,38599623008
WebSockets	Local	0,78378222	5,66	0,13847742402
	Móvel	1,04279889	4,26	0,24478847183
XMPP	Local	0,75758944	5,86	0,12928147440
	Móvel	1,07097056	4,15	0,25806519518
Nenhuma	-	0,09747530	45,55	0,00213996267

6.3.3 Resultados dos modelos

A análise de modelos de ciclo de vida da bateria inclui quatro cenários considerando as condições de conectividade distintas e diferentes protocolos de mensagens.

A Tabela 6.13 apresenta as probabilidades de conectividade para todos os cenários, obtidas a partir da CTMC da Figura 5.11. Como esperado, o cenário #4 apresenta os piores resultados, obtendo uma probabilidade de 0,9069969 (90,69969%) de não estar ligado a nenhuma rede, e apenas 0,0476545 (4,76545%) de estar conectado na rede local, e de 0,0453479 (4,53479%) de estar conectado na rede rede móvel. Os cenários #1 e #2 apresentaram os mesmos valores para a rede local (95,22283%), porque mesmo com uma boa conexão para as duas redes, no cenário #1, a rede local é a preferencial. No entanto, uma diferença na conectividade para os dois primeiros cenários é observada através da probabilidade de o dispositivo não estar ligado a qualquer rede, tal como no cenário #1 é de apenas 0,22794%, e no cenário #2 é de 4,54973%. O cenário #3 tem 90,68863% de chances de estar conectado à rede móvel, contra 4,76545% da rede local, e 4,54586% de não estar conectado em nenhuma rede.

A Tabela 6.14 mostra os resultados para um único ciclo de vida útil da bateria, considerando todos os cenários. Estes resultados foram obtidos a partir do modelo da Figura 5.10,

Tabela 6.13: Resultados do modelo de conectividade

Cenário	Conectividade (P)		
	Rede local	Rede móvel	Nenhuma
#1	0,9522283	0,0454922	0,0022794
#2	0,9522283	0,0022744	0,0454973
#3	0,0476545	0,9068863	0,0454586
#4	0,0476545	0,0453479	0,9069969

que adota os valores das Tabelas 6.12 e 6.13 como parâmetros de entrada. Considerando-se todos os cenários, o protocolo XMPP foi o mais eficiente em três dos quatro cenários, com um ciclo de 5,765 horas no cenário #1, 6,129 horas no cenário #2, e 25,659 horas no cenário #4. O WebSocket foi o segundo melhor protocolo no *ranking* geral, o melhor protocolo no cenário #3, com um ciclo de 4,5165 horas, e o segundo melhor no cenário # 1 (5,589 horas), no cenário #2 (5,921 horas) e no cenário #4 (25,656 horas). O protocolo Long Polling foi o pior em todos os cenários, e o protocolo Polling foi o segundo pior.

Tabela 6.14: Tempos de descarga por ciclo

Cenários	Tempo de descarga (h)			
	Polling	Long Polling	WebSockets	XMPP
#1	5,07671	4,19221	5,58902	5,76491
#2	5,42738	4,42701	5,92155	6,12945
#3	3,59208	3,58569	4,51658	4,40992
#4	23,50144	22,44413	25,65629	25,65971

Agora que sabemos que o tempo médio do ciclo de vida de bateria para cada cenário, podemos estimar a vida útil total da bateria. A Figura 6.18 mostra a vida útil da bateria para todos os cenários.

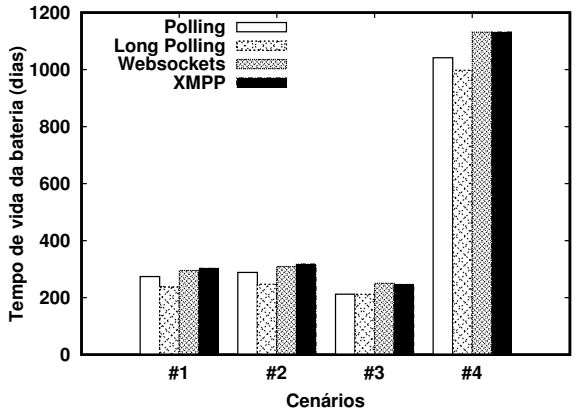


Figura 6.18: Tempo de vida da bateria para todos os cenários

O cenário #4 atingiu o tempo de vida da bateria mais alto, com um mínimo de 997,672 dias para o protocolo Long Polling, e um máximo de 1.131,654 dias para o protocolo XMPP.

O cenário #3 apresenta os menores tempos de vida, com um mínimo de apenas 211,903 dias (Long polling), e um máximo de 250.691 dias (WebSockets). O cenário #2 foi o segundo melhor cenário, seguido pelo cenário #1. Nos cenários #1 e #2, todos os procotocolos obdeceram a um mesmo padrão, tendo o protocolo XMPP alcançado maior tempo de vida, seguido pelos protocolos WebSockets, Polling e Long Polling, respectivamente.

Vale ressaltar que consideramos o tempo de vida da bateria como sendo o tempo médio de descarga de um ciclo de vida útil da bateria, mais o tempo médio para recarregar a bateria, vezes o número de ciclos de vida (ver Equação 5.25). E o tempo de utilização da bateria refere-se à utilização contínua da bateria, isto é, é apenas o tempo médio para a descarga de um ciclo de vida da bateria, vezes o número de ciclos de vida, ou seja, $\Gamma = \phi \times \beta$. A Tabela 6.15 mostra o tempo de utilização da bateria para todos os cenários.

Tabela 6.15: Utilização da bateria para todos os cenários

Cenários	Tempo de utilização (dias)			
	Polling	**Long Polling**	**WebSockets**	**XMPP**
#1	211,5296	174,6754	232,8758	240,2044
#2	226,1408	184,4585	246,7315	255,3937
#3	149,6700	149,4037	188,1911	183,7467
#4	979,2267	935,1721	1069,0121	1069,1546

No entanto, ter um tempo de vida mais longo da bateria não significa que o serviço será prestado corretamente. Especialmente no caso do cenário #4, que alcançou o maior tempo de vida da bateria, mas é o cenário em que ambas as redes de comunicação possuem conectividade ruim. Por isso, também iremos avaliar a disponibilidade do serviço.

A Figura 6.19 mostra os resultados de disponibilidade para todos os cenários.

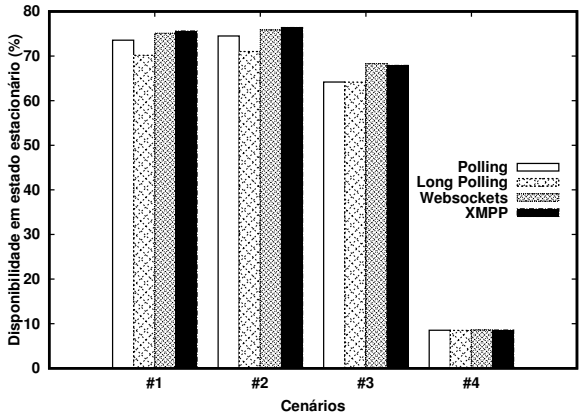


Figura 6.19: Disponibilidade do serviço móvel para todos os cenários

É bastante interessante notar que o cenário #2 apresenta os melhores resultados de disponibilidade, mesmo quando o cenário #1 tem uma boa conectividade para ambas as redes de comunicação, enquanto o cenário #2 tem boa conectividade apenas para a rede local. Isto

pode ser explicado devido ao cenário #2 ter boa conectividade em uma conexão de rede e, especialmente, por ter um tempo de descarga mais longo do que o cenário #1 (ver Tabela 6.14).

No cenário #2, o protocolo XMPP obteve os melhores resultados de disponibilidade (76,386%), seguido pelo protocolo WebSocket que alcançou resultados semelhantes (75,862%), e pelos protocolos Polling (74,491%) e Long Polling (71,016%). O cenário #1 apresenta resultados de disponibilidade muito próximos do cenário #2. Os protocolos de comunicação também seguiram a mesma classificação deste cenário, com 75,611% para XMPP, 75,123% para WebSocket, 73,553% para o Polling, e 70,176% para o Long Polling. Já o cenário #3 apresentou um *ranking* de disponibilidade totalmente diferente para os protocolos de comunicação, tendo o protocolo WebSocket apresentando uma melhor disponibilidade, seguido pelos protocolos XMPP (67,891%), Polling (64,182%), e Long Polling (64,149%), com os dois últimos protocolos tem resultados quase idênticos. Finalmente, observamos que mesmo alcançando um tempo de vida da bateria mais longo, o cenário #4 apresenta os piores resultados de disponibilidade, com apenas 8.568% de disponibilidade para os protocolos XMPP e WebSockets, 8,525% para o Polling, e 8.501% para o Long Pooling.

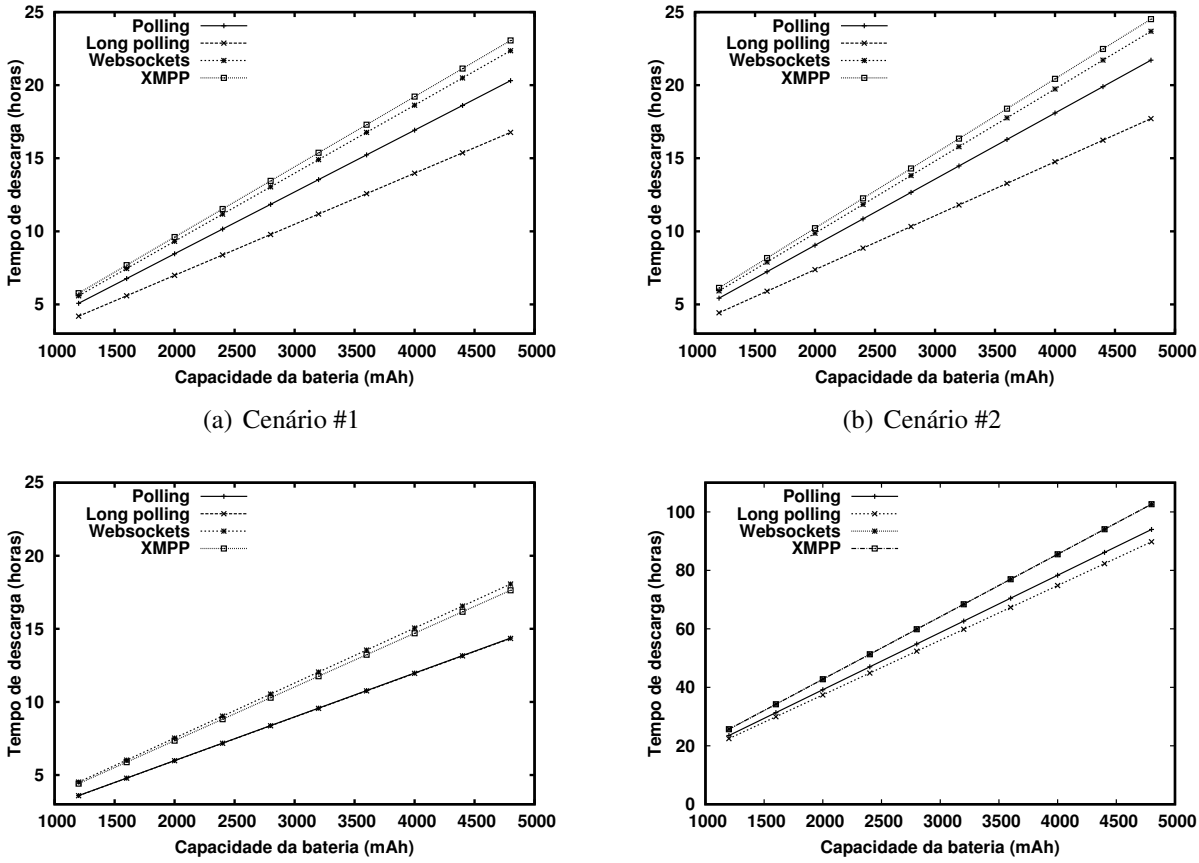
Nós sabemos que os dispositivos móveis com baterias de maior capacidade também podem ter telas maiores, maior poder de processamento, diferentes pacotes de *software* e outros fatores que podem influenciar diretamente o tempo de descarga da bateria. No entanto, sabendo que o dispositivo móvel adotado no experimento possui uma capacidade de bateria de 1200mAh, e considerando que as mesmas condições são mantidas, é possível identificar o impacto que diferentes baterias pode ter sobre o tempo de descarga de cada ciclo. A Figura 6.20 mostra uma variação paramétrica para mostrar a influência de baterias com diferentes capacidades no ciclo de vida.

Uma vez que a vida útil da bateria depende do número de ciclos utilizados, por isso, quando cada ciclo demora mais tempo para ser consumido, a vida útil da bateria aumenta. Portanto, podemos concluir que o investimento em baterias de maior capacidade influencia positivamente a vida útil da bateria em si.

Considerando os sistemas móveis em geral, uma vida útil da bateria curta pode significar a necessidade de substituição de todo o equipamento, uma vez que alguns dispositivos móveis (como *smartphones* e *tablets*) tem a sua bateria integrada ao resto do *hardware*, o que torna difícil de substituir apenas o bateria. Portanto, para este tipo de sistema, o ideal é adotar dispositivos com bateria facilmente substituíveis, tanto por permitir a substituição imediata após um único ciclo de descarga, aumentando a disponibilidade do serviço, bem como evita a necessidade de substituição de todos os equipamentos em longo prazo em razão da morte prematura da bateria.

6.4 Tempo médio de entrega de mensagens em sistemas de MCC

Identificar a disponibilidade e confiabilidade de uma infraestrutura de *mobile cloud computing* pode não ser o suficiente quando se trata de alguns tipos de sistemas, a exemplo



(a) Cenário #1

(b) Cenário #2

(c) Cenário #3

(d) Cenário #4

Figura 6.20: Influência de diferentes capacidades da bateria no ciclo de vida

do *mHealth*, já que não há garantias de que uma mensagem de comunicação enviada a partir de um dos componentes da infraestrutura irá chegar em um tempo satisfatório. Portanto, é necessário identificar o impacto causado por alguns componentes no tempo médio de entrega das mensagens.

Esta seção tem por objetivo identificar o tempo médio de entrega de mensagens de *mobile cloud computing* considerando três conjuntos de cenários distintos, com: diferentes tipos de conectividade das redes sem fio; diferentes taxas de descarga de baterias; e usando diferentes *timeouts*. Os detalhes sobre a arquitetura adotada e os modelos propostos estão descritos na Seção 5.4.

6.4.1 Parâmetros de entrada

A Tabela 6.1 mostra os parâmetros para o modelo do dispositivo móvel. A Tabela 6.2 mostra os parâmetros de entrada para componentes comuns a todos os cenários avaliados pelo modelo de conectividade mostrados na Figura 5.11. Os parâmetros de entrada para os modelos da infraestrutura de nuvem são apresentados na Tabela 6.3. Por fim, a Tabela 6.16 apresenta os parâmetros de entrada para os diferentes tipos de conectividade considerados nesse estudo.

Tabela 6.16: Parâmetros de entrada do modelo de conectividade

Parâmetro	Valor (h^{-1})	Descrição
λ_{rt}	1/10000	Falha do roteador
μ_{rt}	1/12	Reparo do roteador
λ_{tw}	1/83220	Falha da torre de comunicação
μ_{tw}	1/12	Reparo da torre de comunicação
λ_{rl_c}	1/2.777778	Entrada em uma região sem cobertura da rede local
μ_{rl_c}	1/0.277778	Saída da região sem cobertura da rede local
λ_{rm_c}	1/2.777778	Entrada em uma região sem cobertura da rede móvel
μ_{rm_c}	1/0.277778	Saída da região sem cobertura da rede móvel

6.4.2 Resultados dos modelos

A Figura 6.21 mostra os resultados considerando diferentes possibilidades de comunicação: um cenário com apenas a rede local; outra com apenas a rede móvel; outro com ambas as rede (local e móvel); outro cenário com sinal da rede local mais estável que a rede móvel; e, finalmente, a rede móvel mais estável que a rede local. Para um sinal mais estável do que o outro, considerou-se uma taxa duas vezes maior que o cenário de referência (*baseline*). Todos estes cenários estão sendo considerados porque o dispositivo móvel pode ir à diferentes áreas cobertas.

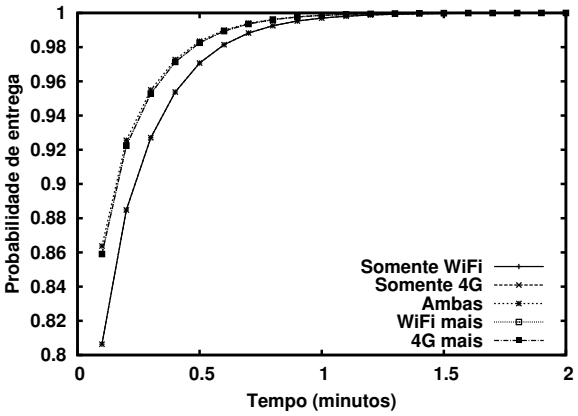


Figura 6.21: Probabilidade de entrega de mensagens com diferentes cenários de conectividade

Os resultados mostram uma sobreposição das linhas da rede local (WiFi) e da rede móvel (4G). Isto significa que, quando é considerado toda a arquitetura de *mobile cloud computing*, estes cenários possuem um comportamento muito parecido. A probabilidade de que uma mensagem seja enviada e imediatamente entregue ao dispositivo móvel é de cerca de 80,63%, e de 99,99% a partir de 1,8 minutos. Um comportamento semelhante pode ser observado quando é considerado que a rede local é mais estável do que a móvel, e quando a rede móvel é mais estável do que a rede local, ou seja, também há uma sobreposição da linhas desses cenários. Agora, é possível observar melhores resultados. A probabilidade de que uma mensagem seja imediatamente entregue é de cerca de 85,90%, e de 99,99% em 1,6 minutos. Finalmente, temos resultados ligeiramente

melhores quando a rede local e a rede móvel estão disponíveis para uso, a probabilidade de entrega no tempo 0 é de 86,37% e 99,99% em 1,6 minutos.

A Figura 6.22 mostra os resultados considerando diferentes taxas de descarga da bateria: 9, 12, 15, 18, 21 e 24 horas. Todos estes cenários estão sendo considerados porque o dispositivo móvel pode ter uma autonomia de bateria diferente, em razão da: adoção de baterias mais potentes; diferentes configurações de *hardware*; perfis distintos de usuário que utilizam funções diferentes do telefone. Nove horas é o pior cenário, quando temos todos os recursos do dispositivo estão sendo utilizados (PHONEARENA, 2015).

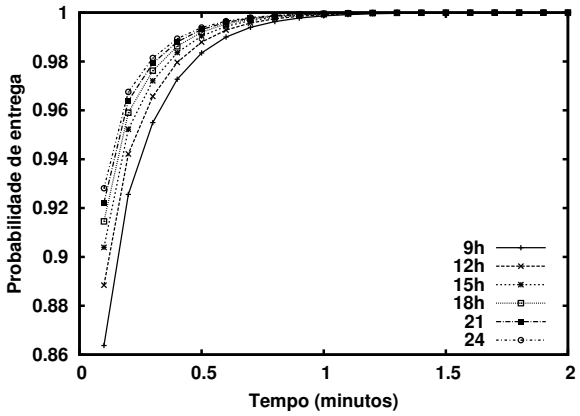


Figura 6.22: Probabilidade de entrega de mensagens com diferentes taxas de descarga da bateria

Como esperado, os piores resultados são obtidos quando temos tempos de descarga mais baixos. Para uma taxa de descarga de 9 horas, a probabilidade de uma mensagem ser enviada e imediatamente entregue ao dispositivo móvel é de cerca de 86,37%, e de 99,99% em 1,6 minutos. Para uma taxa de descarga de 24 horas, a probabilidade de uma mensagem ser entregue imediatamente é de cerca de 92,81% e de 99,99% em 1,3 minutos. Podemos observar uma diferença significativa quando se compara o melhor e o pior cenário. A probabilidade de entrega imediata para as taxas de descarrega 12, 15, 18 e 21 horas são 88,84%, 90,39%, 91,45% e 92,22%; e com 99,99% para a probabilidade, é de 1,5, 1,4, 1,4 e 1,3 minutos, respectivamente.

A Figura 6.23 mostra os resultados considerando diferentes *timeouts*: 10, 20, 30, 40, 50 e 60 segundos. Todos estes cenários estão sendo considerados porque o *timeout* para verificar a entrega de uma mensagem é uma decisão de projeto. Obviamente, quanto menor este tempo, a probabilidade de entrega de uma mensagem será melhor. No entanto, isso pode causar perda de desempenho porque mais mensagens estarão viajando pela rede.

Como esperado, os melhores resultados são obtidos quando adotamos menores *timeouts*. Para um *timeout* de 10 segundos, a probabilidade de uma mensagem a ser enviada e imediatamente entregue ao dispositivo móvel é de cerca de 86,37%, e de 99,99% em 1,6 minutos. Para um *timeout* de 60 segundos, a probabilidade de uma mensagem ser entregue imediatamente é de cerca de 83,57% e 99,99% em 8,8 minutos. Podemos observar uma enorme diferença quando se compara o melhor e o pior cenário. A probabilidade de entrega imediata para o *timeout* de

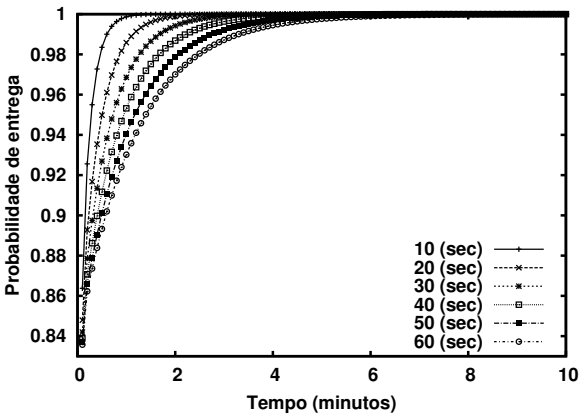


Figura 6.23: Probabilidade de entrega de mensagens com diferentes *timeouts*

20, 30, 40 e 50 segundos são 84,78%, 84,19%, 83,89% e 83,70%; e com 99,99% de chances de entrega, é de 3,0, 4,5, 5,9 e 7,3 minutos, respectivamente.

Com isso, foi possível identificar que o *timeout* é o parâmetro que possui maior impacto sobre a entrega de mensagens do sistema; que as mudanças na comunicação possuem algumas semelhanças; e que baterias com maior capacidade também ajudam a qualidade do serviço oferecido.

6.5 Considerações finais

Este capítulo apresentou a aplicação dos modelos e estratégias propostas em cenários reais, demonstrando a eficácia da abordagem adotada. Os resultados obtidos através dos estudos de caso permitem identificar quais parâmetros/componentes possuem maior influência no funcionamento do sistema como um todo.

7

Conclusões e trabalhos futuros

Você não sabe o quanto eu caminhei pra chegar até aqui!

—CIDADE NEGRA

Esta tese alcançou uma série de resultados nas áreas que explorou, e a maior contribuição é o conjunto de modelos para planejamento de infraestruturas de *mobile cloud computing*. Esta tese também fornece guias para melhorar os sistemas de forma contínua, servindo como suporte em um processo de planejamento da infraestrutura. As técnicas foram testadas por meio de estudos de casos distintos e produziu diversas outras contribuições.

Foram propostos modelos de alto nível que caracterizam o comportamento geral de um sistema de MCC. Para tornar isto possível, foi necessário propor sub-modelos que representavam todos os componentes móveis na nuvem: infraestrutura de nuvem, comunicação sem fio e dispositivos móveis. O modelo de nuvem foi dividido em vários sub-modelos para diminuir a complexidade do modelo global.

Apresentamos diversos estudos de disponibilidade e confiabilidade de infraestruturas de *mobile cloud computing* usando modelos analíticos e hierárquicos. Foi avaliado o impacto dos mecanismos de redundância na dependabilidade do sistema através da combinação de modelos RBD e modelos de recompensa de Markov. Os resultados mostram que os níveis de dependabilidade do sistema podem ser melhorados com a utilização de componentes ou peças redundantes, tais como baterias reserva para dispositivos móveis e servidores de espera para alguns componentes da infraestrutura da nuvem. A modelagem analítica também auxiliou na avaliação dos custos. A variação paramétrica de taxas de falha e de reparo assinalou o impacto dos componentes do sistema nas métricas. Esta análise também mostrou que o custo de implantação e manutenção de uma infraestrutura de nuvem privada para um serviço de nuvem móvel é viável em operações de médio a longo prazo, quando comparado com o custo de alugar uma infraestrutura semelhante em uma nuvem pública. Assim, a decisão de adotar nuvens privadas ou públicas pode variar de acordo com as singularidades das aplicações.

Além disso, analisamos o impacto de efeitos do envelhecimento de *software* e de ações

proativas de rejuvenescimento na disponibilidade e confiabilidade da MCC. Para que isso fosse possível, foi necessário propor uma metodologia para realização de experimentos, que por sua vez demandou a necessidade de implementação de ferramental de apoio e implantação de ambientes de testes. Os resultados obtidos com os experimentos também serviram como parâmetros de entrada para os modelos propostos. O ferramental desenvolvido durante a tese poderá ser utilizado por administradores de sistemas para auxiliar no gerenciamento de recursos da infraestrutura e tomada de decisão.

Esta tese também propõe modelos analíticos para representar a conectividade do telefone móvel e da descarga da bateria em configurações distintas de cobertura de rede e protocolos de comunicação. Esta abordagem de modelagem proporciona meios para analisar o impacto dessas variáveis sobre a vida útil da bateria, ou seja, o tempo total de utilização de uma bateria se todos os ciclos de descarga estão esgotados, exigindo a substituição da bateria. A análise considerou uma arquitetura de computação móvel projetada para o monitoramento remoto de pacientes. Os protocolos WebSockets e XMPP produziram os maiores tempos de vida de bateria. O cenário com cobertura de rede fraca para ambas as tecnologias, locais e móveis, também demonstrou a melhor vida útil da bateria, apesar da péssima qualidade do serviço oferecida. A conectividade ruim economiza bateria porque somente poucos dados são transmitidos de forma eficaz, o que representa, portanto, um risco para em diversos cenários de aplicação, a exemplo de um paciente que está sendo monitorado em um ambiente de *mHealth*. Entre os cenários onde o serviço foi devidamente entregue, o melhor é quando há uma boa cobertura para a conexão de rede local e fraca cobertura para a conexão de rede de área ampla (ou seja, 3G ou 4G).

Além disso, também foi proposto um modelo que representa o caminho percorrido por uma mensagem para ser entregue ao destinatário em um ambiente de *mobile cloud computing*. Foi utilizado o formalismo CTMC para representar toda a complexidade de uma comunicação usando dois tipos de rede sem fio (local e móvel), que podem estar disponível para uso ou não. Os resultados mostraram vários cenários considerando diferentes possibilidades de comunicação, taxas de descarga de bateria e diferentes tempos de espera (*timeouts*). Assim, foi possível identificar que o *timeout* é a métrica que tem um maior impacto na disponibilidade do sistema; que as mudanças na comunicação têm semelhanças; e que baterias mais eficientes também ajudam a melhorar a disponibilidade do sistema.

7.1 Limitações

Mesmo tendo proposto diversos modelos e alcançado vários resultados, obviamente esta tese possui algumas limitações.

A principal limitação é que os modelos da infraestutura de *mobile cloud computing* não são gerados automaticamente. De fato, essa não é uma abordagem trivial, pois seria necessário a implementação de uma aplicação de alto nível que monitorasse as falhas da infraestrutura em longo prazo e armezenasse em um banco de dados para a geração automática dos modelos. Além

disso, esse *framework* deveria ser capaz de interpretar os dados do monitoramento e entender o funcionamento da arquitetura do sistema, os componentes que a integram e suas dependências, para que fosse possível a geração autormática dos modelos. Além disso, ainda seria necessário se comunicar com a API de algum *software* de modelagem que desse suporte ao formalismo adotado, como o Mercury (SILVA et al., 2015), por exemplo.

Uma outra limitação é que a ferramenta de monitoramento desenvolvida e as estratégias de rejuvenescimento de *software* implementadas são aplicáveis somente a sistemas operacionais Linux. De fato, a implementação de mecanismos automáticos de monitoramento para sistemas como Windows e MacOS não foram efetivados em razão de dificuldades de acesso à chamadas de sistema específicas. Além disso, os sistemas Linux foram escolhidos por sua natureza e pela grande quantidade de distruições com código aberto.

7.2 Trabalhos futuros

Embora esta tese tenha alcançado diversos resultados e coberto alguns pontos relacionados ao planejamento de infraestruturas de *mobile cloud computing*, há muitas possibilidade de estender o trabalho atual. Algumas dessas possibilidades podem ser implementados em trabalhos trabalhos futuros, e estão listadas abaixo:

Poderíamos propor modelos mais específicos que representem o comportamento do ambiente, considerando fatores como distância, altitude, intensidade do sinal e até células de comunicação. Esse cenário parece bastante promissor, pois a partir disso seria possível identificar detalhes importantes no comportamento do sistema, como a influência desses fatores na qualidade das redes de comunicação, no atraso da entrega de mensagens e até no atraso/perca de requisições durante a transição do usuário em diferentes cenários. Além disso, também seria possível avaliar métricas como disponibilidade, confiabilidade, desempenho e performabilidade do sistema. Possivelmente seria necessário a execução de experimentos em ambientes controlados para obtenção de alguns dos parâmetros de entrada, mas também é possível obter parte desses dados a partir da literatura, ou ainda a partir de estimativas, já que tais valores dependem de diversas questões de infraestrutura e até geográficas e, portanto, não podem ser considerados somente como valores fixos e arbitrários.

Além disso, também é possível propor um modelo de tempo de vida da bateria que não considere um comportamento linear, mas com uma distribuição que lhe representar com maior similaridade. A ideia inicial é analisar medições reais do tempo de descarga da bateria com diferentes tipos de aplicações, a exemplo dos diferentes protocolos de comunicação adotados nesta tese, e aplicar testes de *Goodness of Fit* (GOF) para criar um *ranking* das funções de distribuição de probabilidade que melhor se ajustam ao comportamento das amostras utilizadas. A partir daí seria possível propor modelos mais próximos ao comportamento real do sistema. Rede de Petri estocástica (SPN) parece ser um dos formalismos mais adequados, pois quando

usado em conjunto com o métódo de *moment-matching* possibilita a representação da distribuição em seus modelos. Assim, será possível identificar com maior exatidão tanto o tempo final de vida da bateria, mas também o comportamento da perda de capacidade com o passar do tempo, auxiliando também na antecipação das aquisição de novos equipamentos.

Também é possível estender os resultados obtidos nesta tese a partir de técnicas de otimização para testar diferentes métricas e cenários. Adotar algoritmos de otimização para esse tipo de cenário é bastante factível, pois é possível trabalhar com uma grande gama de possibilidades e ainda encontrar o melhor cenário para as métricas de interesse estabelecidas. Questões como disponibilidade, confiabilidade e custo, por exemplo, podem ser comparadas para encontrar o cenário que alcance os melhores resultados para cada um deles separadamente, ou até mesmo um resultado que integre todos eles em conjunto. Esse tipo de abordagem também pode ser adotada em conjunto com outros modelos para facilitar a representação do sistema e até mesmo o custo computacional necessário para calcular os resultados. Além disso, técnicas de anáise de sensibilidade como variação paramétrica e derivada parcial também podem ser utilizadas para propor melhorias nos resultados. Dessa maneira, seria possível testar inúmeras novas possibilidade sob uma diferente perspectiva.

Cenários específicos das sub-áreas de *Edge Computing* e *Internet of Things* podem ser avaliados para melhor caracterização do sistema e obtenção de resultados mais precisos. Em ambos os cenários também é possível considerar questões de infraestrutura e de mobilidade do usuário para avaliar o comportamento do sitema. A avaliação de métricas como performabilidade parece ser um bom ponto de partida, pois possibilita identificar o impacto causado no desempenho do sistema por problemas na infraestrutura computacional ou até mesmo por questões geográficas. Estes cenário proporcionam diversas possibilidades de organização e planejamento da infraestrutura, pois possui inúmeros fatores e componentes que podem influenciar em métricas como disponibilidade e confiabilidade.

Referências

ABDALLAH, H.; HAMZA, M. On the sensitivity analysis of the expected accumulated reward. **Performance Evaluation**, Amsterdam, The Netherlands, The Netherlands, v.47, n.2, p.163–179, 2002.

ADB. **Android Debug Bridge (ADB)**. Available in: *http://developer.android.com/tools/help/adb.html*, Android Developers page.

ALONSO, J. et al. A comparative experimental study of software rejuvenation overhead. **Performance Evaluation**, [S.l.], v.70, n.3, p.231–250, 2013.

AMAZON. **Amazon Elastic Block Store (EBS)**. Available in: http://aws.amazon.com/ebs, Amazon.com, Inc.

AMAZON. **Amazon Web Services - Simple Monthly Calculator**. Online. Accessed 10-Jan-2017.

AMORETTI, M.; GRAZIOLI, A.; ZANICHELLI, F. A modeling and simulation framework for mobile cloud computing. **Simulation Modelling Practice and Theory**, [S.l.], v.58, p.140–156, 2015.

ARAUJO, C. et al. Supporting availability evaluation in MCC-based mHealth planning. **Electronics Letters**, [S.l.], v.52, n.20, p.1663–1665, 2016.

ARAUJO, J. et al. Software Rejuvenation in Eucalyptus Cloud Computing Infrastructure: a method based on time series forecasting and multiple thresholds. In: INTERNATIONAL WORKSHOP ON SOFTWARE AGING AND REJUVENATION (WOSAR'11) IN CONJUCTION WITH THE 22ND ANNUAL INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING (ISSRE'11), 3., Hiroshima, Japan. **Proceedings...** [S.l.: s.n.], 2011.

ARAUJO, J. et al. Software Aging Issues on the Eucalyptus Cloud Computing Infrastructure. In: IEEE INT. CONF. ON SYSTEMS, MAN, AND CYBERNETICS (SMC'11), Anchorage, USA. **Proceedings...** [S.l.: s.n.], 2011.

ARAUJO, J. et al. Experimental Evaluation of Software Aging Effects on the Eucalyptus Cloud Computing Infrastructure. In: ACM/IFIP/USENIX INTERNATIONAL MIDDLEWARE CONFERENCE (MIDDLEWARE'11), Lisbon, Portugal. **Proceedings...** [S.l.: s.n.], 2011.

ARAUJO, J. et al. Dependability evaluation of a mhealth system using a mobile cloud infrastructure. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS, SMC 2014, SAN DIEGO, CA, USA, OCTOBER 5-8, 2014, 2014. **Anais...** [S.l.: s.n.], 2014. p.1348–1353.

ARDUINO. **About Arduino**. 2015.

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

AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B. **Fundamental concepts of dependability**. [S.l.]: University of Newcastle upon Tyne, Computing Science, 2001.

BALASUBRAMANIAN, N.; BALASUBRAMANIAN, A.; VENKATARAMANI, A. Energy Consumption in Mobile Phones: a measurement study and implications for network applications. In: ACM SIGCOMM CONFERENCE ON INTERNET MEASUREMENT CONFERENCE, 9., New York, NY, USA. **Proceedings...** ACM, 2009. p.280–293. (IMC '09).

BARABADY, J.; KUMAR, U. Availability allocation through importance measures. **International journal of quality & reliability management**, [S.l.], v.24, n.6, p.643–657, 2007.

BARBERA, M. V. et al. To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In: INFOCOM, 2013 PROCEEDINGS IEEE. **Anais...** [S.l.: s.n.], 2013. p.1285–1293.

BASNEY, J.; LIVNY, M. Managing Network Resources in Condor. In: HPDC. **Anais...** [S.l.: s.n.], 2000. p.298–299.

BATTERYUNIVERSITY. **How to Prolong Lithium-based Batteries**. Available in: http://batteryuniversity.com/learn/article/how_to_prolong_lithium_based_batteries. Accessed in Sep 2015.

BEZERRA, M. C. et al. Availability Modeling and Analysis of a VoD Service for Eucalyptus Platform. **2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)**, [S.l.], p.3779–3784, Outubro 2014.

BIRNBAUM, Z. W. On the importance of different components in a multicomponent system. **Multivariate Analysis**, [S.l.], v.2, 1968.

BLAKE, J. T.; REIBMAN, A. L.; TRIVEDI, K. S. Sensitivity analysis of reliability and performability measures for multiprocessor systems. In: SIGMETRICS '88: PROCEEDINGS OF THE 1988 ACM SIGMETRICS CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, New York, NY, USA. **Anais...** ACM, 1988. p.177–186.

BLUM, R. **Linux Command Line and Shell Scripting Bible**. [S.l.]: Wiley Publishing, Inc, 2008. 840p.

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

BOURDENA, A. et al. Using socio-spatial context in mobile cloud process offloading for energy conservation in wireless devices. **IEEE Trans. Cloud Comput**, [S.l.], v.1, 2015.

CALLOU, G. et al. Models for dependability and sustainability analysis of data center cooling architectures. In: DSN WORKSHOPS. **Anais...** IEEE, 2012. p.1–6.

CALLOU, G. et al. Estimating sustainability impact of high dependable data centers: a comparative study between brazilian and us energy mixes. **Computing**, [S.l.], p.1–34, 2013.

CHUNG, Y. W. An improved energy saving scheme for instant messaging services. **Wireless Advanced (WiAd), 2011**, London, UK, p.278–282, 2011.

CISCO. **Cisco Visual Networking Index**: global mobile data traffic forecast update, 2015–2020. [S.l.]: White Paper, 2016.

CLOTH, L.; JONGERDEN, M. R.; HAVERKORT, B. R. Computing Battery Lifetime Distributions. In: THE 37TH ANNUAL IEEE/IFIP INT. CONF. ON DEPENDABLE SYSTEMS AND NETWORKS (DSN 2007). **Anais...** IEEE Computer Society, 2007. p.780–789.

COOPER, T.; FARRELL, R. Value-chain engineering of a tower-top cellular base station system. In: VEHICULAR TECHNOLOGY CONFERENCE, 2007. VTC2007-SPRING. IEEE 65TH. **Anais...** [S.l.: s.n.], 2007. p.3184–3188.

COTRONEO, D. et al. Software Aging Analysis of the Linux Operating System. In: SOFTWARE RELIABILITY ENGINEERING (ISSRE), 2010 IEEE 21ST INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2010. p.71–80.

COTRONEO, D. et al. A Survey of Software Aging and Rejuvenation Studies. **J. Emerg. Technol. Comput. Syst.**, New York, NY, USA, v.10, n.1, p.8:1–8:34, Jan. 2014.

COTRONEO, D.; ORLANDO, S.; RUSSO, S. Characterizing Aging Phenomena of the Java Virtual Machine. In: RELIABLE DISTRIBUTED SYSTEMS, 2007. SRDS 2007. 26TH IEEE INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2007. p.127–136.

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

D-LINK Wireless N150 Router. Available in: `http://www.dlink.com/us/en/home-solutions/connect/routers/dir-601-wireless-n-150-home-router`. Online. Acessed 02-March-2016.

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

DELL. **Datacenter capacity planner**. Online. Acessed 21-March-2016.

DELL. **DELL computers**. Online. Acessed 21-March-2016.

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.

DISTERER, G.; KLEINER, C. {BYOD} Bring Your Own Device. **Procedia Technology**, [S.l.], v.9, p.43 – 53, 2013. {CENTERIS} 2013 - Conference on {ENTERprise} Information Systems / ProjMAN 2013 - International Conference on Project MANagement/ {HCIST} 2013 - International Conference on Health and Social Care Information Systems and Technologies.

EC2. **Amazon Elastic Compute Cloud - EC2**. Online. Acessed 01-March-2016, Amazon.com, Inc.

ENERGY, U. D. of. **City of Palo Alto utilities - Palo Alto clean (clean local energy acessible now)**. Online. Acessed 22-May-2013, `http://energy.gov`.

EUCALYPTUS SYSTEMS, I. **Eucalyptus Open-Source Cloud Computing Infrastructure - An Overview**. 130 Castilian Drive, Goleta, CA 93117 USA: Eucalyptus Systems, Inc., 2009.

FERRO, E.; POTORTI, F. Bluetooth and Wi-Fi wireless protocols: a survey and a comparison. **Wireless Communications, IEEE**, [S.l.], v.12, n.1, p.12–26, 2005.

FOURSQUARE. Available in: *https://foursquare.com/*, Foursquare.com, Inc.

FOWLER, M. **UML distilled**: a brief guide to the standard object modeling language. [S.l.]: Addison-Wesley Professional, 2004.

FRIEDMAN, R.; KOGAN, A.; KRIVOLAPOV, Y. On Power and Throughput Tradeoffs of WiFi and Bluetooth in Smartphones. **IEEE Transactions on Mobile Computing**, Piscataway, NJ, USA, v.12, n.7, p.1363–1376, July 2013.

GRAY, J.; REUTER, A. **Transaction processing**. [S.l.]: Morgan Kaufíann Publishers, 1993.

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

GROTTKE, M.; NIKORA, A. P.; TRIVEDI, K. S. An empirical investigation of fault types in space mission system software. In: DEPENDABLE SYSTEMS AND NETWORKS (DSN), 2010 IEEE/IFIP INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.447–456.

GROTTKE, M.; TRIVEDI, K. S. Fighting bugs: remove, retry, replicate, and rejuvenate. **Computer**, [S.l.], v.40, n.2, p.107–109, 2007.

GUIMARAES, A. et al. Impact analysis of availability on computer networks infrastructure. In: COMMUNICATION SOFTWARE AND NETWORKS (ICCSN), 2011 IEEE 3RD INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.171–175.

GUIMARãES, A. P.; MACIEL, P. R. M.; JR., R. M. An analytical modeling framework to evaluate converged networks through business-oriented metrics. **Reliability Engineering and System Safety**, [S.l.], p.81–92, 2013.

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.

HAPNER, M. et al. Java Message Service. **Sun Microsystems Inc., Santa Clara, CA**, [S.l.], 2002.

HAVERKORT, B. R. **Markovian models for performance and dependability evaluation**. New York, NY, USA: Springer-Verlag New York, Inc., 2002. p.38–83.

HOFFMAN, F.; GARDNER, R. Evaluation of Uncertainties in Environmental Radiological Assessment Models. In: TILL, J.; MEYER, H. (Ed.). **Radiological Assessments**: a textbook on environmental dose assessment. Washington, DC: U.S. Nuclear Regulatory Commission, 1983. Report No. NUREG/CR-3332.

HUANG, Y. et al. Software rejuvenation: analysis, module and applications. In: SYMP. ON FAULT TOLERANT COMPUTING, FTCS-25, 25., Pasadena. **Proceedings...** [S.l.: s.n.], 1995. p.381–390.

HYNDMAN, R. J.; ATHANASOPOULOS, G. **Forecasting**: principles and practice. [S.l.]: OTexts, 2014.

ITANI, W.; KAYSSI, A.; CHEHAB, A. Energy-efficient incremental integrity for securing storage in mobile cloud computing. In: INTERNATIONAL CONFERENCE ON ENERGY AWARE COMPUTING, 2010. **Anais...** [S.l.: s.n.], 2010. p.1–2.

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

KAYS, J.; REHTANZ, C. Planning process for distribution grids based on flexibly generated time series considering RES, DSM and storages. **IET Generation, Transmission & Distribution**, [S.l.], v.10, n.14, p.3405–3412, 2016.

KIM, D. S.; GHOSH, R.; TRIVEDI, K. S. A Hierarchical Model for Reliability Analysis of Sensor Networks. **Pacific Rim International Symposium on Dependable Computing, IEEE**, Los Alamitos, CA, USA, v.0, p.247–248, 2010.

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.

KOLMOGOROV, A. Über die analytischen Methoden in der Wahrscheinlichkeitsrechnung (in German). **Mathematische Annalen**, [S.l.], 1931. Springer-Verlag.

KOUTRAS, V.; SALAGARAS, C.-P.; PLATIS, A. Software Rejuvenation for Higher Levels of VoIP Availability and Mean Time to Failure. In: DEPENDABILITY OF COMPUTER SYSTEMS, 2009. DEPCOS-RELCOMEX '09. FOURTH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2009. p.99–106.

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

LAPRIE, J. C.; AVIZIENIS, A.; KOPETZ, H. (Ed.). **Fault Tolerance**: principles and practice. 2nd.ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1990.

LIN, Y. et al. Performance evaluation of remote display access for mobile cloud computing. **Computer Communications**, [S.l.], v.72, p.17–25, 2015.

LOGOTHETIS, D.; TRIVEDI, K. Time-dependent behavior of redundant systems with deterministic repair. In: **Computations with Markov Chains**. [S.l.]: Springer, 1995. p.135–150.

LOMOTEY, R. K.; DETERS, R. Mobile-based medical data accessibility in mHealth. In: MOBILE CLOUD COMPUTING, SERVICES, AND ENGINEERING (MOBILECLOUD), 2014 2ND IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014. p.91–100.

MACIEL, P. et al. Dependability Modeling. **In Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions**, Hershey, PA, p.53 – 97, 2012.

MALHOTRA, M. Power-Hierarchy of Dependability Model Types. **IEEE Trans. on Reliability**, [S.l.], v.43, n.2, p.493–502, Sept. 1994.

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.

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

MATIAS, R.; Freitas Filho, P. J. An Experimental Study on Software Aging and Rejuvenation in Web Servers. In: ANNUAL INT. COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE (COMPSAC'06), 30., Chicago. **Proceedings...** [S.l.: s.n.], 2006.

Matos Junior, R. et al. Experimental Evaluation of Software Aging Effects in the Eucalyptus Elastic Block Storage. In: IEEE INT. CONF. ON SYSTEMS, MAN, AND CYBERNETICS (SMC'12), Seoul, Korea. **Proceedings...** [S.l.: s.n.], 2012.

MATOS JUNIOR, R. et al. Software Rejuvenation in Eucalyptus Cloud Computing Infrastructure: a hybrid method based on multiple thresholds and time series prediction. **International Transactions on Systems Science and Applications**, [S.l.], v.8, p.1–16, 2012.

MATOS, R. et al. Sensitivity analysis of a hierarchical model of mobile cloud computing. **Simulation Modelling Practice and Theory**, [S.l.], v.50, n.0, p.151–164, 2015.

MCC Forum. **Mobile Cloud Computing Forum**. Available in: `http://www.mobilecloudcomputingforum.com/`.

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

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

MELO, M. et al. Comparative Analysis of Migration-Based Rejuvenation Schedules on Cloud Availability. In: SYSTEMS, MAN, AND CYBERNETICS (SMC), 2013 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2013. p.4110–4115.

MONKEY. **UIApplication Exerciser Monkey**. Available in: *http://developer.android.com/tools/help/monkey.html*, Android Developers page.

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

MUPPALA, J. K.; TRIVEDI, K. S. GSPN models: sensitivity analysis and applications. In: ACM-SE 28: PROCEEDINGS OF THE 28TH ANNUAL SOUTHEAST REGIONAL CONFERENCE, New York, NY, USA. **Anais...** ACM, 1990. p.25–33.

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

OLIVEIRA, D. et al. Availability and Energy Consumption Analysis of Mobile Cloud Environments. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS, MANCHESTER, SMC 2013, UNITED KINGDOM, OCTOBER 13-16, 2013. **Anais...** [S.l.: s.n.], 2013. p.4086–4091.

OMIDI, A.; MORADI, S. Modeling and Quantitative Evaluation of an Internet Voting System Based on Dependable Web Services. In: COMPUTER AND COMMUNICATION ENGINEERING (ICCCE), 2012 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2012. p.825–829.

PANNEERSELVAM, J. et al. Mobilouds: an energy efficient mcc collaborative framework with extended mobile participation for next generation networks. **IEEE Access**, [S.l.], 2016.

PARK, J. et al. Markov chain based monitoring service for fault tolerance in mobile cloud computing. In: ADVANCED INFORMATION NETWORKING AND APPLICATIONS (WAINA), 2011 IEEE WORKSHOPS OF INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.520–525.

PENG, J.; WU, Q. Research and Implementation of RSA Algorithm in Java. **Management of e-Commerce and e-Government, International Conference on**, Los Alamitos, CA, USA, v.0, p.359–363, 2008.

PHONEARENA. **Phone Arena - Phone News, Reviews and Specs**. Online. Acessed 10-Oct-2015, `http://www.phonearena.com/`.

POUSHTER, J. **Smartphone Ownership and Internet Usage Continues to Climb in Emerging Economies**: but advanced economies still have higher rates of technology use. Available in: http://www.pewglobal.org/2016/02/22/smartphone-ownership-and-internet-usage-continues-to-climb-in-emerging-economies/. Accessed in December 2016.

QURESHI, S. S. et al. Mobile cloud computing as future for mobile applications - Implementation methods and challenging issues. In: CLOUD COMPUTING AND INTELLIGENCE SYSTEMS (CCIS), 2011 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** IEEE, 2011. p.467–471.

RAUSAND, M.; HØYLAND, A. **System reliability theory**: models, statistical methods, and applications. [S.l.]: John Wiley & Sons, 2004. v.396.

REITER, A.; ZEFFERER, T. Flexible and Secure Resource Sharing for Mobile Augmentation Systems. In: IEEE INTERNATIONAL CONFERENCE ON MOBILE CLOUD COMPUTING, SERVICES, AND ENGINEERING (MOBILECLOUD), 2016. **Anais...** [S.l.: s.n.], 2016. p.31–40.

ROYCE, W. W. Managing the development of large software systems. In: IEEE WESCON. **Anais...** [S.l.: s.n.], 1970. v.26, n.8.

SALIH, R. M.; OTHMANE, L. B.; LESZEK, L. Privacy Protection in Pervasive Healthcare Monitoring Systems with Active Bundles. **Parallel and Distributed Processing with Applications Workshops (ISPAW), 2011 Ninth IEEE International Symposium on**, [S.l.], p.311–315, 2011.

SATYANARAYANAN, M. et al. The case for vm-based cloudlets in mobile computing. **IEEE Pervasive Computing**, [S.l.], v.8, n.4, p.14–23, 2009.

SCHMIDT, D. C. et al. **Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects**. [S.l.]: John Wiley & Sons, 2013. v.2.

SCHMIDT, M. **Depreciation Expense and Depreciation Schedules Explained**: definitions, meaning, and example calculations. [Online; accessed 02-March-2015], https://www.business-case-analysis.com/depreciation.html.

SCHROEDER, B.; GIBSON, G. A. Disk failures in the real world: what does an mttf of 1,000,000 hours mean to you? In: USENIX CONFERENCE ON FILE AND STORAGE TECHNOLOGIES, 5., Berkeley. **Proceedings...** [S.l.: s.n.], 2007. (FAST '07).

SHIRAZ, M. et al. Energy efficient computational offloading framework for mobile cloud computing. **Journal of Grid Computing**, [S.l.], v.13, n.1, p.1–18, 2015.

SHOOMAN, M. L. **Reliability of Computer Systems and Networks**: fault tolerance, analysis, and design. New York, NY, USA: John Wiley & Sons, Inc., 2002.

SILVA, B. et al. Mercury: an integrated environment for performance and dependability evaluation of general systems. **In: Proceedings of Industrial Track at 45th Dependable Systems and Networks Conference (DSN-2015)**, [S.l.], 2015.

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

SQUARETRADE. **Cell Phone Comparison Study**. Online. Accessed 01-March-2015, http://www.squaretrade.com/cell-phone-comparison-study-nov-10.

SUDHA, G.; GANESAN, R. Secure transmission medical data for pervasive healthcare system using android. **Communications and Signal Processing (ICCSP), 2013 International Conference on**, [S.l.], p.433–436, 2013.

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

THEIN, T.; PARK, J. S. Availability analysis of application servers using software rejuvenation and virtualization. **Journal of computer science and technology**, [S.l.], v.24, n.2, p.339–346, 2009.

TRIVEDI, K. et al. Modeling High Availability. In: DEPENDABLE COMPUTING, 2006. PRDC '06. 12TH PACIFIC RIM INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2006. p.154–164.

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.; VAIDYANATHAN, K.; GOŠEVA-POPSTOJANOVA, K. Modeling and analysis of software aging and rejuvenation. In: SIMULATION SYMPOSIUM, 2000.(SS 2000) PROCEEDINGS. 33RD ANNUAL. **Anais...** [S.l.: s.n.], 2000. p.270–279.

ULLRICH, M.; LäSSIG, J.; GAEDKE, M. Towards Efficient Resource Management in Cloud Computing: a survey. In: IEEE 4TH INTERNATIONAL CONFERENCE ON FUTURE INTERNET OF THINGS AND CLOUD (FICLOUD), 2016. **Anais...** [S.l.: s.n.], 2016. p.170–177.

VAIDYANATHAN, K.; TRIVEDI, K. S. Extended classification of software faults based on aging. In: FAST ABSTRACT, INT. SYMP. SOFTWARE RELIABILITY ENG., HONG KONG. **Anais...** [S.l.: s.n.], 2001.

VAIDYANATHAN, K.; TRIVEDI, K. S. A Comprehensive Model for Software Rejuvenation. **IEEE Transactions on Dependable and Secure Computing**, Los Alamitos, v.2, p.124–137, April 2005.

VALLINA-RODRIGUEZ, N.; CROWCROFT, J. Energy Management Techniques in Modern Mobile Handsets. **Communications Surveys & Tutorials, IEEE**, [S.l.], v.15, p.179–198, 2012.

WANG, S.-L. et al. Design and evaluation of a cloud-based Mobile Health Information Recommendation system on wireless sensor networks. **Computers & Electrical Engineering**, [S.l.], v.49, p.221–235, 2016.

WEI, B.; LIN, C.; KONG, X. Dependability modeling and analysis for the virtual clusters. In: COMPUTER SCIENCE AND NETWORK TECHNOLOGY (ICCSNT), 2011 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. v.4, p.2316–2320.

WINSTON, W. L.; GOLDBERG, J. B. Operations research: applications and algorithms. In: **Anais...** Duxbury press Belmont: CA, 1994.

XIANG, M.; TAUCH, S.; LIU, W. Dependability and Resource Optimation Analysis for Smart Grid Communication Networks. In: BIG DATA AND CLOUD COMPUTING (BDCLOUD), 2014 IEEE FOURTH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014. p.676–681.

Apêndice

A

Planejamento dos experimentos de envelhecimento e rejuvenescimento de *software*

A execução de experimentos para investigar a ocorrência do fenômeno do envelhecimento de *software* necessita da implementação de dois requisitos principais: técnica de monitoramento e estratégia de geração de carga de trabalho. Portanto, esta subseção apresenta separadamente os detalhes do planejamento dos experimentos de envelhecimento de *software* realizados no dispositivo móvel e na infraestrutura de computação em nuvem.

A.1 Dispositivo móvel

Para o monitoramento dos recursos do dispositivo móvel foram utilizados *scripts* Bash comuns em sistemas Linux. No entanto, mesmo o Android sendo um sistema operacional baseado em Linux, muitos programas nativos do Linux não foram incluídos nele, e, portanto, a coleta de dados teve de ser simplificada. Para a proposta do presente experimento, o comando *top* (BLUM, 2008) foi utilizado para reunir medidas de consumo de recursos, tais como a utilização de memória.

Um computador foi utilizado para coletar informações e gerar a carga de trabalho. A comunicação entre o computador e o *smartphone* foi mediada através do *Android Debug Bridge* (ADB) (ADB, 2015). Esta é uma ferramenta de linha de comando que permite um computador controlar um dispositivo Android através de uma ligação USB. O ADB é uma aplicação cliente-servidor que inclui três componentes: *server*, *client* e *daemon*. O *server* é executado como um processo em segundo plano na máquina *desktop* e gerencia a comunicação entre o *client* e o *adb daemon* em execução no dispositivo. O *client* é invocado a partir de um comando shell *adb*, enquanto o daemon é executado como um processo em segundo plano em cada instância do dispositivo (ADB, 2015).

Um ambiente de testes foi configurado para executar os experimentos (ver Figura A.1). Um computador foi utilizado para simular as solicitações dos usuários e o comportamento resultante da carga de trabalho foi observado em um smartphone. O *script* responsável por

monitorar todos os aplicativos em execução no dispositivo é descrito abaixo:

```
$ adb -s <serialNumber> shell “top">> top-log.txt
```

onde, $ representa o *prompt* de linha de comando; *adb* inicia uma comunicação ADB; *-s <serialNumber>* envia um comando ADB a uma instância específica do dispositivo, referenciado pelo seu número de série atribuído pelo ADB; *shell* inicia um comando shell remoto na instância alvo do dispositivo; *top* é o comando da aplicação que coleta as informações sobre a utilização de recursos do dispositivo móvel; o símbolo >> adiciona a saída de dados para um arquivo específico; *top-log.txt* é o arquivo texto onde todas as informações serão salvas.

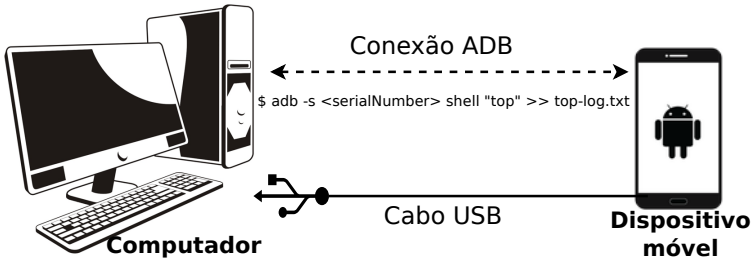


Figura A.1: Arquitetura do ambiente de testes para o dispositivo móvel

Os dados foram coletados a cada 10 segundos executando o *script* descrito acima. A frequência de amostragem foi escolhida como um compromisso entre a exigência de observar todos os eventos importantes, evitando interferência do mecanismo de monitoramento no comportamento do sistema.

Para investigar os efeitos do envelhecimento de *software*, como o vazamento de memória em aplicativos do Android, o dispositivo móvel teve de ser monitorado em condições de funcionamento intensas, as quais estaria sujeito a falhas. Monitorar o ambiente sob condições normais e simplesmente esperar que os efeitos do envelhecimento apareçam pode levar dias, meses ou até mesmo anos, e mesmo assim ainda pode nunca sequer ocorrer (ARAUJO et al., 2011). Portanto, é necessário gerar uma carga de trabalho capaz de estressar o sistema, acelerando o processo de falha. Para o propósito desse estudo, foi adotada a aplicação *Exerciser Monkey* (MONKEY, 2015). O *Monkey* simula toques aleatórios, cliques e outros eventos do usuário, e é comumente usado por designers Android para aplicações na fase de desenvolvimento/teste.

Se executado sem especificar as opções, o *Monkey* inicia em um modo silencioso (não detalhado), e envia eventos aleatórios para todos os pacotes instalados no alvo (MONKEY, 2015). Uma linha de comando típica do *Monkey* é apresentada abaixo:

```
$ adb -s <serialNumber> shell monkey -p <packageName> 100
```

O *Monkey* utiliza a ferramenta ADB para enviar comandos para um dispositivo especificado através de um prompt. O comando *-s <serialNumber>* contém o número de série do dispositivo. O comando *-p <packageName>* especifica o pacote a ser estressado, e 100 é o número de eventos pseudo-aleatórios enviados pelo *Monkey*.

Um *script* Bash foi criado para automatizar a geração da carga de trabalho, e definido para ser executado em intervalos de 1 segundo. Deve-se afirmar que a carga de trabalho foi projetada para acelerar as possíveis falhas e ocorrência de sintomas de envelhecimento, e os valores de tempo escolhidos não foram feitos para imitar as condições reais de uso.

A.2 Computação em nuvem

Dada a complexidade no monitoramento e interpretação dos dados da grande quantidade de recursos a serem monitorados no ambiente de nuvem privada, bem como a implementação de estratégias a serem aplicadas em ações de pervenção à falhas, houve a necessidade de criar um ferramental que auxiliasse em tais atividades. O ferramental desenvolvido está descrito em detalhes nos apêndices C e D.

Essa subseção apresenta os detalhes da realização dos experimentos de investigação de envelhecimento de *software*, e o método de rejuvenescimento adotado.

A.2.1 Investigação de envelhecimento de *software*

Com o objetivo de analisar possíveis efeitos de envelhecimento de *software* em plataformas de nuvem privada, implantamos um ambiente de testes composto por seis computadores Core 2 Quad (2,66 GHz, 4 GB de RAM). Cinco deles executando o sistema operacional Linux Ubuntu Server 11.04 (kernel 2.6.38-8 x86-64) e o *framework* Eucalyptus 2.0.2. Um computador foi usado como cliente da nuvem, executando o Ubuntu Desktop 11.04 (kernel 2.6.38-8 x86-64). O sistema operacional em execução nas máquinas virtuais é uma versão personalizada do Ubuntu 9.04, que executa um servidor HTTP simples. O ambiente de nuvem em teste é totalmente baseado no *framework* Eucalyptus e no *hypervisor* KVM. Analisamos a utilização dos recursos de *hardware* e *software* em um cenário em que algumas operações de nuvem relacionadas com a instanciação de máquinas virtuais são continuamente realizadas ao longo do período de aproximadamente 30 dias. A definição da duração do experimento foi baseada na observação empírica até a manifestação de alguns efeitos de envelhecimento, considerando a carga de trabalho que foi adotada. A seguir, descrevemos os principais componentes do nosso ambiente de testes, bem como a carga de trabalho adotada.

A plataforma de computação em nuvem Eucalyptus é composta por cinco componentes de alto-nível: *Cloud Controller* (CLC), *Cluster Controller* (CC), *Storage Controller* (SC), *Node Controller* (NC) e *Walrus*.

- O ***Cloud Controller*** é o *front-end* para toda a infraestrutura de nuvem. Ele é responsável pelo gerenciamento dos recursos virtuais subjacentes (servidores, rede e armazenamento) por meio da API Amazon EC2 (SUN, 2009). Este componente usa interfaces de serviços web para receber as solicitações de ferramentas de cliente em um lado e interagir com o resto dos componentes do Eucalyptus no outro lado;

- O ***Cluster Controller*** coleta informações sobre um conjunto de VMs e agendam a execução da VM em NCs específicos. Ele possui três funções principais: programar solicitações de execução de instâncias de entrada para NCs específicos, controlar a sobreposição de rede virtual de instância e reunir/relatar informações sobre um conjunto de NCs (EUCALYPTUS SYSTEMS, 2009);

- O ***Node Controller*** é executado em cada nó e controla o ciclo de vida das instâncias em execução no nó. O NC interage com o sistema operacional e com o virtualizador sendo executado no nó. Os NCs controlam a execução, inspeção e encerramento de instâncias no host onde ele é executado. Um NC faz consultas para descobrir os recursos físicos do nó, bem como para aprender sobre o estado das instâncias de VM nesse nó (EUCALYPTUS SYSTEMS, 2009; D et al., 2010).

- ***Storage Controller*** provê armazenamento em bloco para uso das instâncias das máquinas virtuais. Ele implementa o armazenamento em bloco acessado pela rede, simular ao ofertado pelo serviço de *Elastic Block Storage* (EBS) da Amazon (AMAZON, 2016), e fornece interface para vários tipos de sistemas de armazenamento (NFS, iSCSI, etc.).

- ***Walrus*** é um serviço de armazenamento de dados baseado em arquivos, que é uma interface compatível com o serviço *Simple Storage Service* (S3) da Amazon. Os usuários que tem acesso ao Eucalyptus podem usar o *Walrus* para enviar/receber dados da nuvem bem como de instâncias executando nos nós. Além disso, ele também atua como serviço de armazenamento das *Virtual Machine* (VM)s;

A Figura A.2 apresenta os componentes principais do ambiente de testes. Os módulos de gerenciamento do Eucalyptus *Cloud Controller, Cluster Controller, Storage Controller* e *Walrus* foram instalados no mesmo computador (host 1), e as máquinas virtuais foram instanciadas em quatro máquinas físicas (hosts 2, 3, 4 e 5), de modo que cada uma delas executa um *Node Controller*. Dessas quatro máquinas, três delas usam um sistema operacional com arquitetura de 32 bits (i386), e uma delas com arquitetura de 64 bits (amd64). Tal estratégia foi adotada para permitir a captura de possíveis diferentes efeitos de envelhecimento de *software* relacionados à arquitetura do sistema. Cada host tem capacidade para executar quatro máquinas virtuais, e cada uma delas é configurada com 1 CPU, 256 MB de RAM e 5 GB de disco. O host *Client* foi usado para monitorar o ambiente, e também para efetuar requisições no *Cloud Controller*, atuando como cliente da infraestrutura nesse *testbed*. Todos os computadores foram interligados em uma rede privada por meio de um *switch*.

A aceleração do tempo de vida do ambiente se deu através da geração de um *workload* específico baseada no ciclo de vida das máquinas virtuais. O referido ciclo de vida é baseado em quatro estados: *Pendente, Executando, Desligando* e *Finalizada*, como apresentado na Figura A.3. *Scripts* Bash foram criados para iniciar, reiniciar, e finalizar as máquinas virtuais

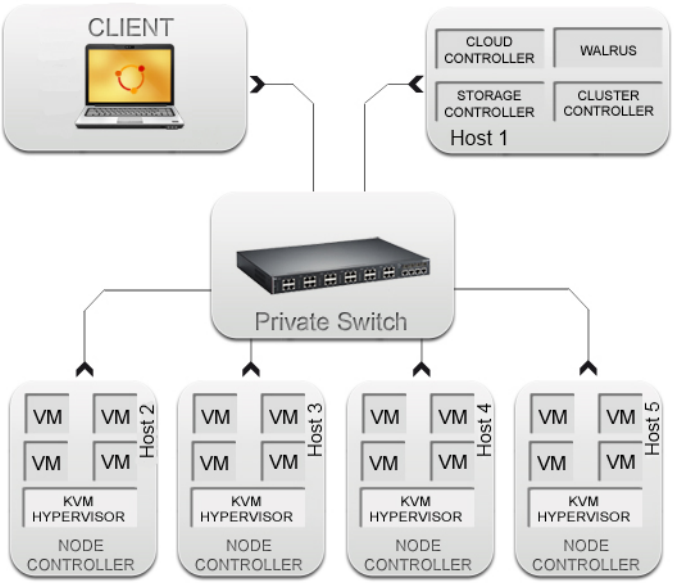


Figura A.2: Componentes do ambiente de testes da nuvem privada

em um curto período de tempo. Tais operações são essenciais para este tipo de ambiente, pois permitem um rápido dimensionamento da capacidade, tanto para mais quanto para menos, bem como quando os requisitos computacionais mudam (comportamento comum na computação elástica) (EC2, 2016).

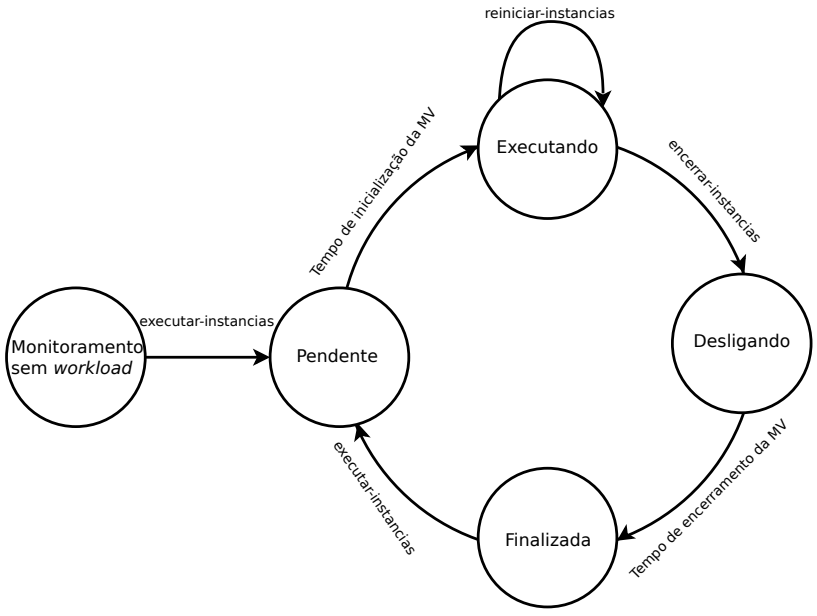


Figura A.3: *Workload* de gerenciamento do ciclo de vida das máquinas virtuais

Aplicativos desenvolvidos para ambientes de nuvem se adaptam ao aumento da demanda instanciando novas máquinas virtuais, e economiza recursos finalizando MVs subutilizadas quando a carga de requisições está baixa. Reinicialização de máquinas virtuais também é essencial para mecanismos de alta disponibilidade que reiniciam MVs automaticamente em outros servidores físicos quando uma falha no servidor é detectada. Nossa carga de trabalho é

implementada por meio de funções de *script* Bash que executam as operações que acabamos de mencionar, como pode ser visto abaixo:

- **Função Instanciar**: Esta função instancia 8 máquinas virtuais em um *cluster*. As MVs são instâncias de um servidor *http*;
- **Função Finalizar**: Esta função identifica quais instâncias estão executando na nuvem e finaliza todas elas;
- **Função Reiniciar**: Muito parecido com a função anterior, ela também identifica todas as instâncias existentes, mas em vez de encerrá-las, esta função executa sua reinicialização.

A cada dois minutos o *script* verifica se já se passaram mais de duas horas a partir do início da última inicialização. Se assim for, todas as máquinas virtuais serão finalizadas, caso contrário, todas as MVs são reinicializadas.

O ambiente é monitorado durante cerca de duas horas sem carga de trabalho. Em seguida, o *script* do *workload* instancia todas as MVs e segue o ciclo de carga de trabalho descrito anteriormente. O período de monitoramento sem executar qualquer carga de trabalho foi escolhido para identificar a relação entre os dados obtidos com e sem carga de trabalho.

A.2.2 Método de rejuvenescimento proposto

Nessa subseção descrevemos o método automatizado adotado para acionar um mecanismo de rejuvenescimento em ambientes de nuvem privada que utilizam a plataforma de computação em nuvem Eucalyptus. O nosso método baseia-se no mecanismo de rejuvenescimento apresentado por (MATIAS; Freitas Filho, 2006), que envia um sinal (SIGUSR1) para o processo mestre do Apache, para que todos os processos filhos ociosos sejam encerrados e novos processos sejam criados. Esta ação limpa a memória acumulada e tem um pequeno impacto sobre o serviço, uma vez que o processo mestre aguarda as conexões estabelecidas serem encerradas.

Num ambiente de produção, o intervalo em que o processo é reiniciado deve ser tão grande quanto possível, a fim de reduzir os efeitos da soma dos pequenos *downtimes* durante um grande período de tempo de execução. Uma abordagem para alcançar esse intervalo máximo é baseado numa alta frequência de monitoramento do recurso a ser analisado, como a utilização de memória de um processo específico. No momento exato em que o limite de memória for alcançado, o rejuvenescimento é acionado. Um pequeno intervalo de amostragem pode afetar o desempenho do sistema, por isso consideramos que um intervalo de 1 minuto é a duração de tempo mínimo para evitar interferências no sistema. Apesar da capacidade para proporcionar bons resultados, pode-se identificar um problema de tal abordagem. É possível que o processo monitorado atinja o seu limite de memória entre dois pontos de monitoramento. Um tempo de inatividade adicional é introduzido desta forma, que vamos descrever como "*downtime* causado pelo monitor".

O mecanismo de acionamento do rejuvenescimento proposto pretende remover este tempo de inatividade adicional, uma vez que tenta manter o intervalo da reinicialização do processo tão grande quanto possível. A previsão sobre quando a UMC vai ser alcançada é usada para essa finalidade. Portanto, considerando-se a classificação das estratégias de rejuvenescimento apresentado na Figura A.4, a nossa abordagem tem características de dois tipos, uma vez que é um rejuvenescimento baseado em *threshold*, mas é auxiliado por predições.

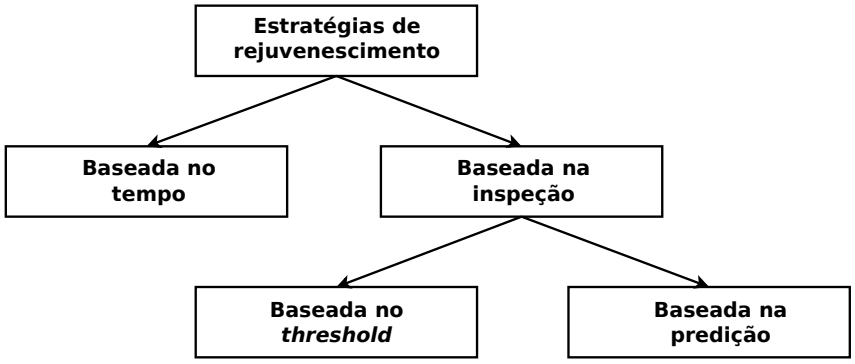


Figura A.4: Estratégias de classificação de rejuvenescimento

Séries temporais apropriadas nos permite realizar uma análise de tendências e, portanto, o tempo restante até que o processo atinja a UMC com um erro aceitável. Esta informação torna possível o agendamento do rejuvenescimento a um determinado momento (T_{rej}), que leva em conta um limite de segurança (T_{safe}) para completar o processo de rejuvenescimento antes da UMC ser atingida. O limite de segurança deve abranger o tempo gasto durante o rejuvenescimento e o tempo relativo ao erro de previsão de séries temporais. Portanto, podemos afirmar qur $T_{rej} = T_{UMC} - T_{safe} = T_{UMC} - (T_{restart} + T_{PredError})$.

Como pode ser visto na Figura A.5, é importante destacar que a análise de tendência só é iniciada após o *script* de monitoramento detectar o crescimento de utilização de memória do processo a partir do ponto de partida do cálculo de séries termporais (CST_{PP}), que no nosso caso é de 80% da utilização da memória crítica. Este ponto de partida foi adotado para evitar interferências desnecessárias no sistema devido ao cálculo das séries temporais. Quando um limite de 95% de UMC é atingido, a última predição gerada pela análise de tendência é utilizada para programar a ação rejuvenescimento, isto é, o último T_{UMC} calculado vai ser utilizado para avaliar o T_{rej} e o sistema será preparado para que o rejuvenescimento ocorra normalmente no tempo T_{rej}. Vemos que para chegar a este ponto final de computação da série temporal (T_{PF}), não há nenhum benefício em continuar computando novas estimativas, e seria um risco adiar o agendamento do mecanismo de rejuvenescimento. É importante salientar que os valores adotados por CST_{PF} e CST_{PP} são específicos para o nosso ambiente e, portanto, pode variar se esta estratégia for adotada em outros tipos de sistemas.

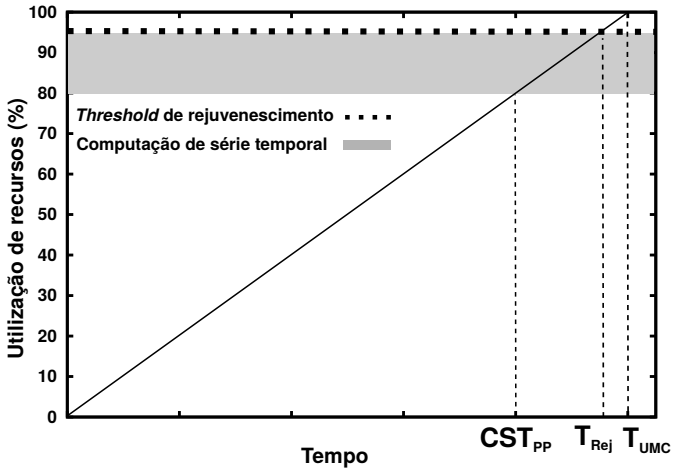


Figura A.5: Projeção da estratégia de rejuvenescimento proposta

B

Planejamento dos experimentos de consumo de energia em dispositivos móveis

Este apêndice tem por objetivo descrever os detalhes relacionados aos experimentos de consumo de energia em dispositivos móveis levando em consideração quatro protocolos de troca de mensagens. Portanto, para que os modelos propostos anteriormente possam ser analisados, precisamos identificar as taxas de descarga dos canais de comunicação durante a utilização contínua de uma rede local (λ_{rl}), rede móvel (λ_{rm}) e sem estar conectado a nenhuma rede (λ_{none}).

Desta maneira, a fim de avaliar os protocolos em termos de consumo de energia, foi desenvolvido um aplicativo de troca de mensagens assíncronas que usa uma arquitetura com base no padrão de publicação-assinatura, no qual um servidor fornece canais para os clientes se conectarem e receber mensagens. O aplicativo inclui quatro diferentes tipos de canais de comunicação correspondentes às estratégias a serem avaliadas: *short polling*, *Comet (long-polling)*, *WebSocket* e *XMPP*. A Figura B.1 ilustra a arquitetura do aplicativo.

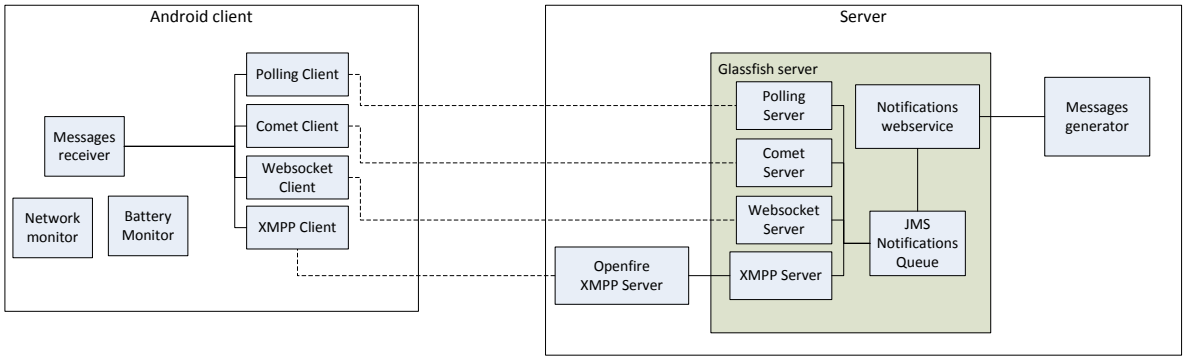


Figura B.1: Arquitetura da aplicação

O sistema de troca de mensagens é baseado no padrão *Java Message Service* (JMS) (HAPNER et al., 2002) e é executado em um servidor GlassFish. As mensagens são publicadaos através de uma chamada de serviço web implementada pelo módulo *Web Service Notifications*. Os módulos *Polling Server*, *Comet Server*, *WebSocket Server* e *XMPP Server* são os canais que os clientes podem se conectar e receber mensagens. Cada módulo possui um lado cliente

correspondente aos protocolos de troca de mensagens. Os módulos são implementados como serviços de *background*.

A fim de efetuar a medição do consumo de energia no dispositivo móvel, foi utilizado uma placa Arduino UNO R3 (ARDUINO, 2015). Este *hardware* permite monitorar a tensão usada em tempo de execução do aplicativo. O dispositivo móvel adotado é um Samsung Galaxy Mini, modelo GT-S5570B, com sistema operacional Android versão 2.3.6 Gingerbread, e capacidade da bateria de 1200 mAh.

A Figura B.2 ilustra o ambiente de testes usado durante a execução dos experimentos. O dispositivo móvel foi conectado à placa Arduíno com a ajuda de um protótipo de placa e a resistência. Para a coleta da potência instantânea usamos um aplicativo de leitura de tensão analógica. Para cada experimento, o aplicativo móvel foi executado com um protocolo de envio de mensagens combinado com um tipo de comunicação, e foi medido o consumo instantâneo da corrente no dispositivo móvel. Uma fonte de alimentação DC foi usada para evitar oscilações na tensão de alimentação para o dispositivo móvel, permitindo obter medições mais precisas. A coleta de dados ocorreu em intervalos de 0,5 segundos.

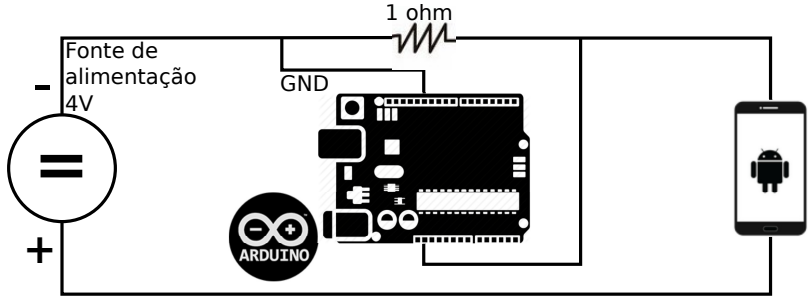


Figura B.2: Visão geral do ambiente de testes

Para entender o consumo de energia de um dispositivo móvel, é necessário entender a Equação B.1. Esta relação define a taxa de consumo de energia de um circuito elétrico.

$$P(t) = V(t) \times I(t) \tag{B.1}$$

onde, V é a potência elétrica (voltagem) em volts, e I é a corrente elétrica em amperes.

A fim de calcular o consumo de energia elétrica E, foi utilizada a Equação B.2, que relaciona a energia P obtida com a equação anterior, com um intervalo de tempo t. O valor obtido representa o consumo de energia elétrica na unidade de Watt-hora dentro de um intervalo de tempo especificado.

$$E = \int_0^t P(t) \times dt \tag{B.2}$$

Através da integração numérica é possível medir o consumo total de energia de cada cenário em nossos experimentos. Ao obter amostras da energia P, em intervalos de 0,5 segundos recolhidos pela placa Arduino, nós aplicamos a Equação B.3 sobre essas amostras para calcular

o total de energia consumida durante o intervalo especificado. Com este método, podemos compreender o consumo de energia durante o experimento e prever a autonomia da aplicação quando se combinam redes locais ou móveis com cada protocolo servidor *push*. Para obter maior precisão, cada experimento foi realizado várias vezes para se obter as diferenças no consumo de energia entre os mesmos cenários.

$$F(x) = \int_0^{\infty} f(x).dx \simeq \sum_{k=0}^{n} f(x_k).t \tag{B.3}$$

Este estudo não considera a linearidade sobre o consumo da bateria, nem leva em conta o consumo de outros aplicativos no dispositivo móvel. Diversos aplicativos do dispositivo móvel foram desinstalados. No entanto, algumas aplicações do sistema, tais como serviços do sistema operacional não puderam ser desinstaladas ou desativadas. Portanto, nos experimentos, o dispositivo móvel operou com aplicações mínimas possíveis, a fim de evitar influências externas de tráfego de dados e uso da CPU causado por esses aplicativos.

C

Ferramenta DR Monitor

Este apêndice apresenta em detalhes a ferramenta *Distributed Resources Monitor* (DR Monitor), fornecendo informações sobre sua arquitetura, recursos e detalhes importantes sobre os módulos de comunicação.

A ferramenta DR Monitor foi desenvolvida para automatizar as atividades de monitoramento em sistemas distribuídos, e tem funções específicas para ambientes de nuvem privada. Ela é capaz de coletar dados sobre a utilização de recursos de todo o sistema a partir de uma determinada máquina (por exemplo, a utilização da CPU, uso de memória RAM, disco, largura de banda, etc.). A ferramenta também monitora o consumo de recursos por aplicações específicas. É possível também identificar alguns sintomas do envelhecimento *software* avaliando parâmetros informados pelo usuário e compará-los com os dados recolhidos em tempo real. Além disso, a ferramenta possibilita configurar e realizar ações básicas de rejuvenescimento. A Figura C.1 mostra a tela principal da ferramenta DR Monitor, onde o usuário pode acessar os recursos de monitoramento, as características do gerador gráfico, ou o manual da ferramenta.

C.1 Arquitetura

A solução apresentada aqui é baseada em uma arquitetura cliente-servidor, onde um nó - chamado de Nó Central (NC) - efetua requisições para um ou mais nós da rede, estes por sua vez são chamados de Nó Distribuído (ND)s. Ambos os componentes têm suporte para comunicação TCP e UDP. A Figura C.2 mostra os componentes e suas interações dentro da arquitetura adotada.

O NC é responsável por identificar, registrar e gerenciar os NDs. Uma mensagem *broadcast* é enviada através da rede, e os NDs ativos respondem com uma mensagem de registro para o NC. Tais atividades são realizadas através da troca de mensagens UDP. Ao receber um pedido de registo, o NC identifica o nome e endereço IP do ND que pediu o registo, e adiciona a uma lista. A partir deste momento, a comunicação entre o NC e ND é executada usando TCP. O NC envia comandos para coleta de utilização de recursosos dos NDs, armazena tais dados recebidos dos NDs, processa essa informação para a visualização do usuário, e identifica ações que possam ser tomadas com base em parâmetros configurados.

Figura C.1: Tela principal do DR Monitor

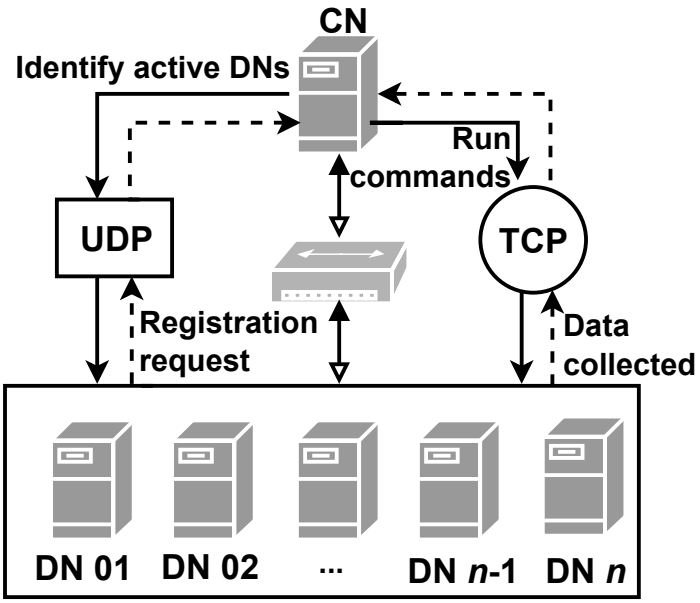


Figura C.2: Arquitetura do DR Monitor

Apenas os NDs que seguem o formato da mensagem UDP do DR Monitor pode ser registrado e enviar dados para o NC, de modo que qualquer outra mensagem com um formato diferente do esperado é descartada.

C.2 Funcionalidades

Esta ferramenta tem várias características que podem ser usadas em qualquer sistema distribuído baseado em Linux, e algumas específicas para ambientes de computação em nuvem privada. Uma delas é permitir o acompanhamento em tempo real da utilização de recursos de máquinas da rede. O monitoramento pode ser realizado de duas maneiras: local ou distribuído. O monitoramento local recolhe dados apenas a partir da máquina NC e salva em arquivos de log. O monitoramento distribuído coleta dados de NDs ativos na rede e salva os dados na máquina NC. Uma visão geral da tela de monitoramento DR Monitor pode ser visto na Figura C.3, que mostra um gráfico único para a métrica de utilização da CPU em nível de usuário (CPU-USER). Esta métrica pertence ao conjunto de recursos gerais, monitorados pelo ND no hospedeiro chamado *jean-note*.

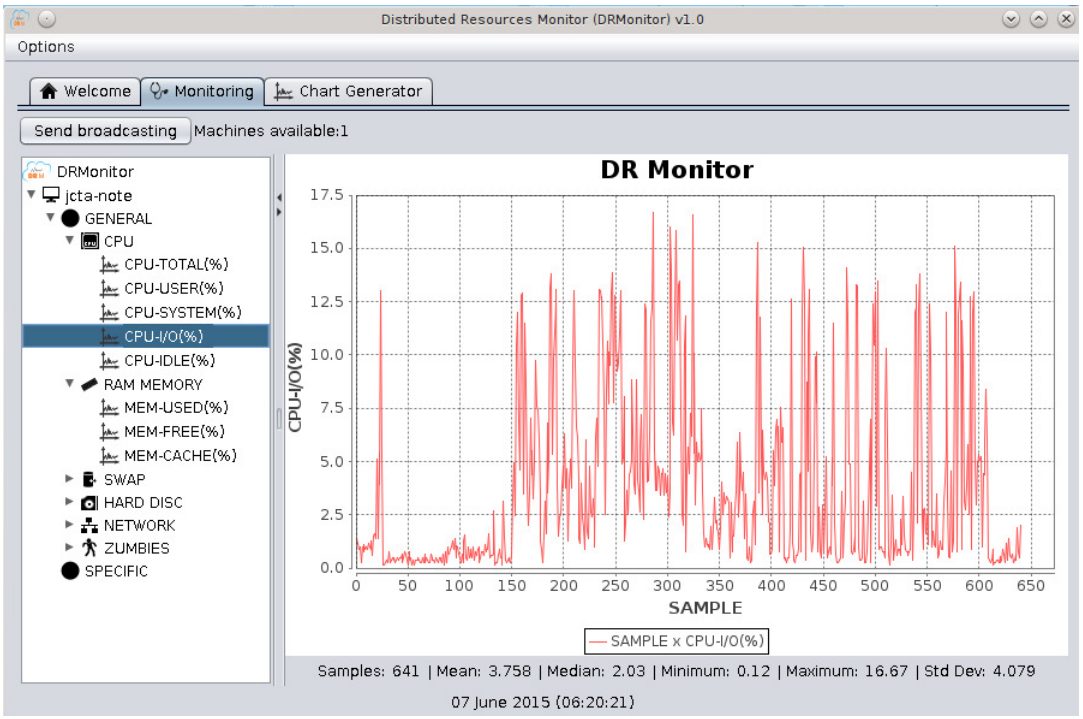


Figura C.3: Tela de monitoramento - gráfico único

A Figura C.4 mostra vários gráficos de monitoramento para todas as métricas relacionadas com a utilização da CPU. Esta funcionalidade multi-gráfico permite ao usuário analisar muitos recursos ou métricas de uma só vez.

No lado esquerdo da Figura C.3 e da Figura C.4, é possível ver os componentes monitorados organizados em uma estrutura de árvore. Esta organização hierárquica auxilia a associação de componentes controlados com as respectivas máquinas. O lado direito mostra um gráfico do componente que foi selecionado na árvore. A parte inferior da janela tem um breve resumo estatístico dos dados apresentados no gráfico.

O DR Monitor é capaz de gerar gráficos *off-line*, como linhas, histograma, dispersão, pizza, box-plot, e gráficos de barras. A Figura C.5 mostra um exemplo de análise *off-line* usando

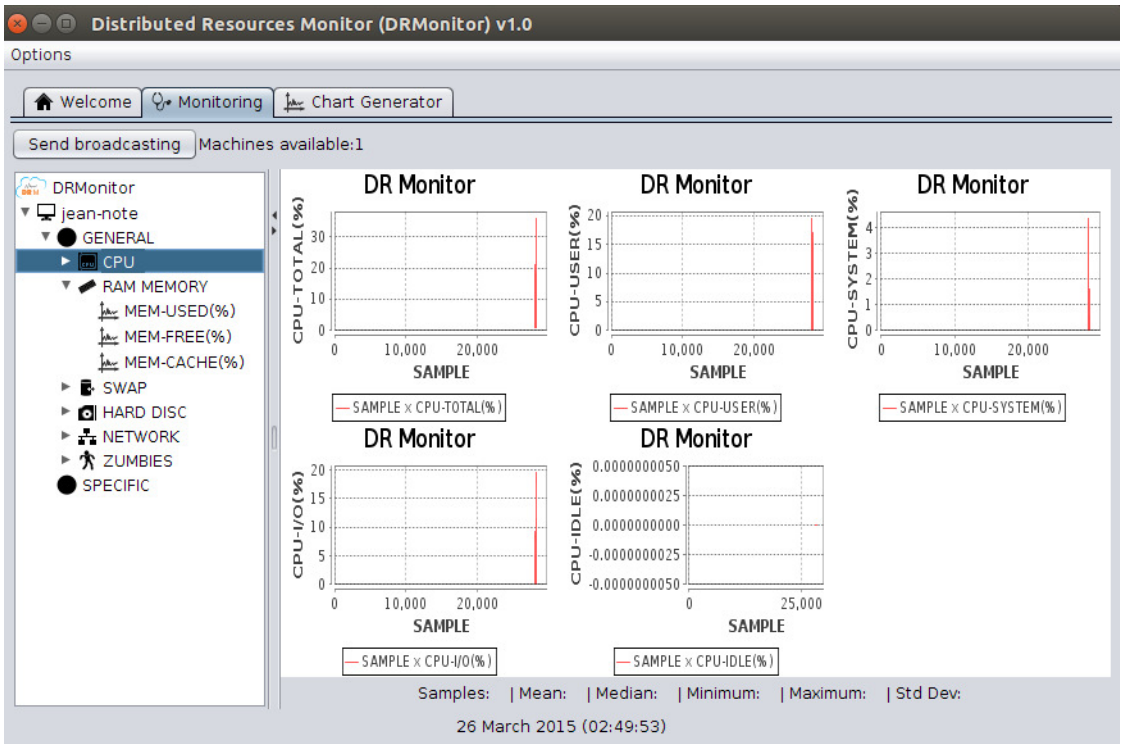


Figura C.4: Tela de monitoramento - multi-gráfico

um gráfico de dispersão.

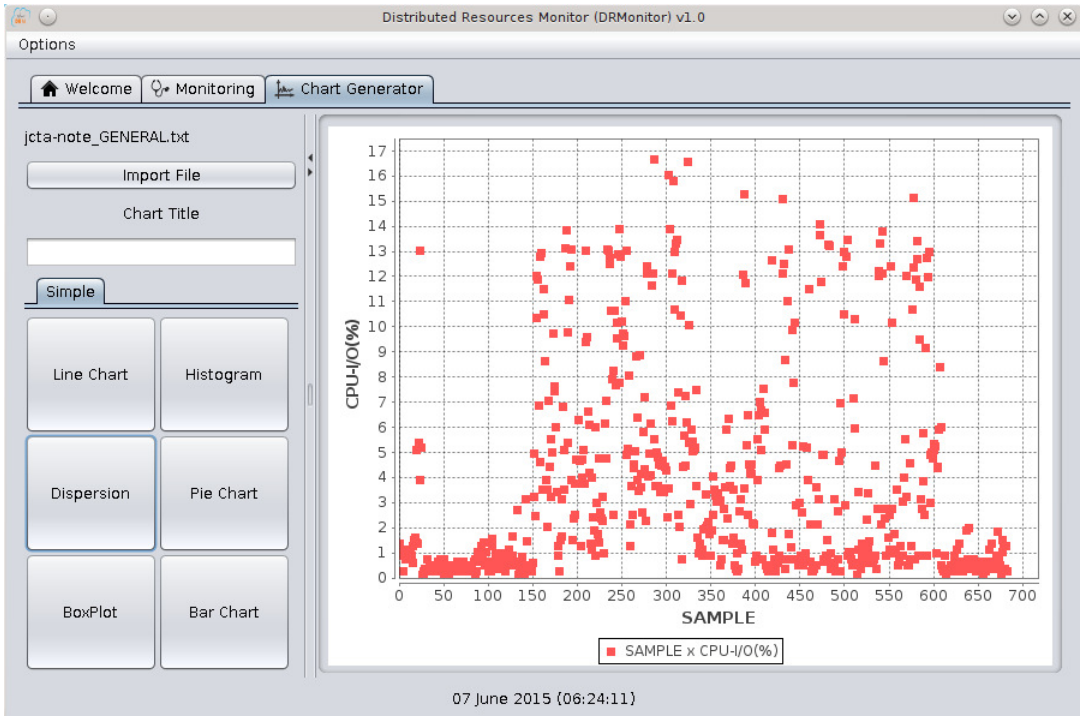


Figura C.5: Análise *off-line* - Gráfico de dispersão

O DR monitor classifica os dados monitorados em dois tipos: gerais ou específicos. O tipo geral refere-se à utilização de recursos de todo o sistema, tais como a utilização da CPU, memória, disco, tráfego de rede e número de processos zumbis. Quando o usuário começa a

monitorar uma máquina, os recursos gerais são solicitados por padrão em intervalos de tempo definidos pelo usuário. O tipo específico refere-se a qualquer processo em execução na máquina monitorada. O usuário escolhe quais processos serão monitorados. As principais informações sobre o consumo de recursos do processo são coletadas, tais como a utilização da CPU, memória virtual, memória residente, e memória total.

Os processos disponíveis na máquina a ser monitorado são listados na tabela que contém o número de identificação do processo (PID) e o seu nome. A Figura C.6 mostra a janela com esta tabela. É permitida a escolha de um ou mais processos, marcando a caixa de seleção na primeira coluna da tabela. É possível ajustar o intervalo de monitoramento e aplicar um filtro digitando parte do nome do processo desejado.

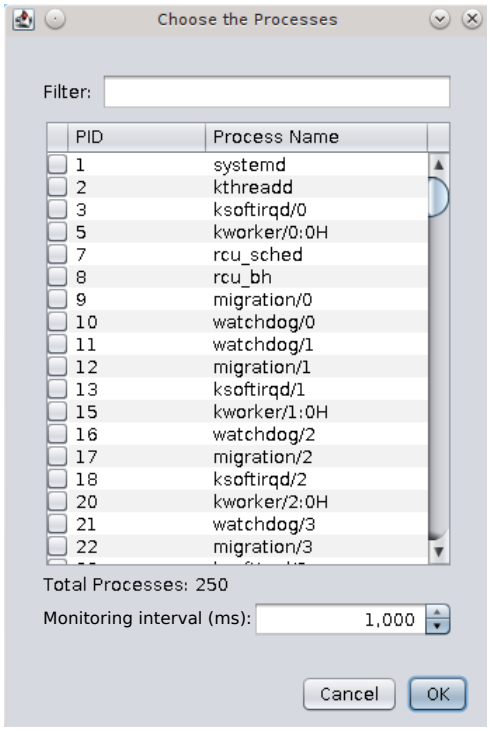


Figura C.6: Seleção do processo

A visualização de dados ocorre em tempo real assim que o NC recebe a informação a partir do respectivo ND. O usuário pode visualizar os dados monitorados através de gráficos de linha, basta clicar sobre o nome do recurso desejado. O gráfico padrão contém uma única série de dados. A ferramenta também permite a geração de gráficos *off-line*, que é uma característica independente de monitoramento em tempo real. Portanto, é possível analisar os dados de registos de monitoramento anteriores, ou qualquer outro conjunto de dados estruturados no formato reconhecido pelo DR Monitor.

É também possível definir um aviso sobre o consumo de recursos, tanto em recursos gerais quanto em processos específicos. A Figura C.7 ilustra um exemplo de aviso.

As informações estatísticas são calculadas automaticamente quando qualquer gráfico é gerado. As medidas estatísticas disponíveis são: número de amostras, média aritmética,

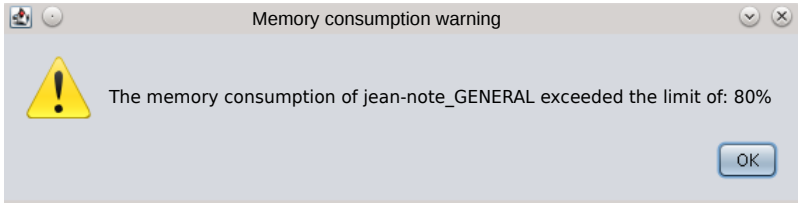


Figura C.7: Exemplo de aviso de consumo de memória

mediana, o maior e o menor valor entre as amostras, e o desvio-padrão. Todas essas medidas são atualizados em tempo real.

A grande vantagem desta ferramenta entre as outras é que além de monitorar diversos recursos em uma interface amigável, ela também implementa algumas ações proativas relacionadas principalemente ao rejuvenescimento de *software*. A nossa abordagem tem características de três categorias: rejuvenescimento baseados no tempo, onde a ação de rejuvenescimento é acionado periodicamente; rejuvenescimento baseado no limiar, onde ação de rejuvenescimento é acionada quando um limiar foi atingido; ou de forma híbrida, impulsionada por aquele que vem em primeiro lugar.

A reinicialização da máquina foi implementada como uma ação geral rejuvenescimento. No entanto, como a maior parte dos serviços web que gerem as plataformas de computação em nuvem são processos Apache, desenvolvemos uma ação de rejuvenescimento específica para estes processos. Esta ação usa o sinal Linux "USR1". O NC ordena que o DN execute o comando "kill -SIGUSR1 $pid", onde a variável "$pid" é o PID do processo Apache alvo. Este sinal pede gentilmente que todos os processos filhos sejam encerrados, e novos processos são criados (MATIAS; Freitas Filho, 2006; ARAUJO et al., 2011). Essa ação libera recursos utilizados pelos processos filhos afetados pelo envelhecimento de *software*, enquanto traz novos processos para atender novos pedidos recebidos.

É importante destacar que a ação de rejuvenescimento para o processo Apache não provoca a interrupção dos serviços em execução no servidor. Uma vez que este comando chega, o sistema operacional encerra o processo filho Apache, mas espera enquanto as conexões ativas são terminadas. Depois disso, cria um novo processo filho para substituir o que foi encerrado, e transfere as novas requisições de serviço para o novo processo. Este procedimento pode causar alguns atrasos no processamento de pedidos recebidos durante a execução da ação de rejuvenescimento, mas não causa indisponibilidade do serviço (MATIAS; Freitas Filho, 2006).

Todas as ações de rejuvenescimento são operações que necessitam de privilégios no sistema operacional da máquina de destino. Portanto, o usuário deve digitar a senha de um usuário com privilégios de administrador, se ele quiser usar os recursos de rejuvenescimento de DR Monitor. Uma vez que esta é uma informação sensível, e precisa ser enviada através da rede de computadores, a senha do usuário é criptografada. A criptografia adotada nesta ferramenta usa a implementação padrão do algoritmo RSA de 1024 bits (PENG; WU, 2008).

D

Ferramenta Predictor

Neste apêndice será descrita a ferramenta Predictor, desde sua concepção até a implementação. Para alcançar o objetivo geral da ferramenta, que é proporcionar um ambiente de predição de alto nível para que projetistas e administradores de sistemas não necessitem de um aprofundamento nas técnicas de predição, foi necessário fazer algumas decisões relacionadas à estrutura do desenvolvimento do projeto, como por exemplo os padrões de projetos utilizados. O fluxo de como as predições são geradas é mostrado na Figura D.1.

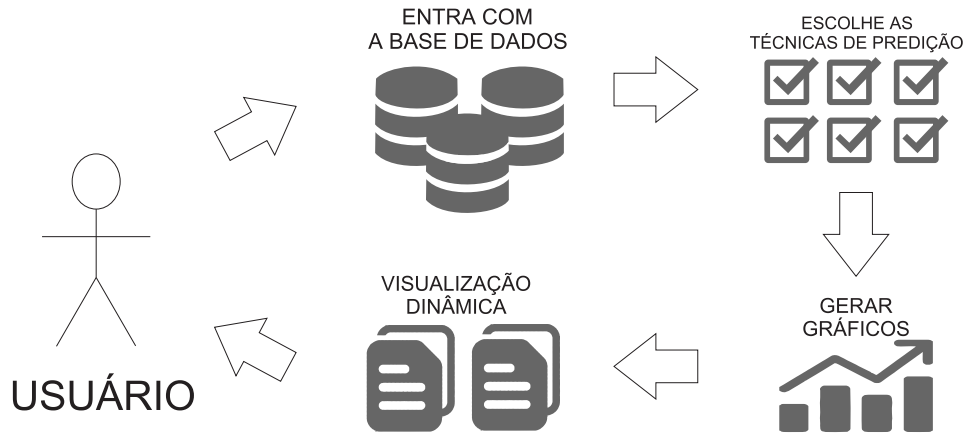


Figura D.1: Fluxo da informação no sistema.

Cada técnica em série temporal pode aumentar ou diminuir a eficácia em base de dados diferentes, ou seja, uma técnica pode ser a melhor para uma determinada base e não ser tão boa assim para outra base. Isto leva a um problema de como saber qual técnica é melhor para uma situação específica.

A ferramenta foi desenvolvida usando a linguagem orientada a objetos JAVA e a linguagem R, onde o usuário deve entrar com uma base de dados usando uma interface amigável, e a ferramenta envia os dados para o pacote R, o qual possui várias técnicas estatísticas de predição de séries temporais. Após a entrada dos dados, o usuário escolhe quais técnicas ele deseja utilizar. A ferramenta dividirá a base em dois conjuntos, um de treino e um de teste, os quais terão 80% para trieino e 20% da base para teste, pois essa divisão é comumente utilizada na literatura (HYNDMAN; ATHANASOPOULOS, 2014). Por fim, a ferramenta escolhe qual das

técnicas possui a melhor acurácia baseado nos menores resultados das métricas *Mean Absolute Error* (MAE), que significa Erro Médio Absoluto, e o *Mean Absolute Percentual Error* (MAPE), que significa Erro Percentual Médio Absoluto. Logo depois desses procedimentos, os gráficos com os treinos e com o resultado final ficam disponíveis para a visualização do usuário.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|f_i - y_i| \tag{D.1}$$

em que f_i é a predição, y_i é o valor atual e n é o número de pontos de predição

$$MAPE = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{A_i - F_i}{A_i}\right| \tag{D.2}$$

em que A_i é o valor atual, F_i é o valor de predição e n é o número de pontos de predição.

Vale ressaltar que a quantidade de amostras numa base não necessariamente implica em uma melhor predição, pelo contrário, o excesso de amostras pode atrapalhar a qualidade da predição (HYNDMAN; ATHANASOPOULOS, 2014), além de aumentar o tempo de processamento na execução das técnicas. Apesar da proposta sugerida envolver várias etapas, o tempo de resposta esperado é curto.

A aplicação desenvolvida é dita como uma aplicação *desktop* e para ser executada necessita tanto do JAVA instalado na máquina, bem como a ferramenta R. Porém, um *script Bash* para o sistema operacional Ubuntu 16.04 foi escrito para instalar o JAVA, o R e os pacotes necessários para que a ferramenta funcione adequadamente.

Os pacotes utilizados e suas dependências são: rJava; zoo; lmtest; quadprog; tseries; timeDate; fracdiff; Rcpp; colorspace; RcppArmadillo; digest; gtable; plyr; stringi; magrittr; RColorBrewer; dichromat; munsell; labeling; scales; stringr; reshape2; ggplot; forecast; expsmooth; fma; fpp. A quantidade de pacotes é grande devido às dependências dos pacotes utilizados no *script* R. Para que o Predictor funcione corretamente, todos esses pacotes devem estar instalados na máquina e na ordem que foram apresentados.

A ferramenta foi desenvolvida utilizando o padrão de projeto *Model View Controller* (MVC) com FACADE, devido a simplicidade e organização desses padrões o que facilita que outros desenvolvedores que estejam interessados em continuar ou modificar esse projeto entendam rapidamente o seu funcionamento. O padrão MVC possui três objetos, que são o controlador (*Controller*) que interpreta as entradas do usuário e as converte em comandos que são enviados para o objeto modelo (*Model*) e, por fim, para a janela de visualização (*View*). O modelo é o responsável por gerenciar a comunicação entre o controlador e a janela de visualização. O padrão FACADE foi escolhido por ser um padrão estrutural, uma vez que define uma forma de criação de relações entre as classes ou entidades. Esse padrão é utilizado para definir uma interface simplificada para um subsistema mais complexo. O padrão de FACADE é ideal quando se trabalha com um grande número de classes interdependentes, ou com as classes que exigem o uso de vários métodos, particularmente quando eles são complicados de usar ou difícil de

compreender. A classe FACADE é um *wrapper* que contém um conjunto de membros que são de fácil compreensão e simples de usar. Esses membros podem acessar o subsistema através da classe FACADE, ocultando os detalhes de implementação (SCHMIDT et al., 2013).

D.1 Modelagem do *software*

Os modelos desenvolvidos durante a modelagem servem para comunicar a estrutura e o comportamento desejados do sistema. Esses modelos podem ter uma visão mais específica, que irá mostrar com mais detalhes o sistema, ou modelos mais abrangentes, os quais revelarão uma visão geral do sistema. Todos os *softwares* podem ser descritos por modelos sob diferentes aspectos, e cada modelo representará uma abstração distinta e específica do sistema. Os modelos classificam-se em dois tipos: os estruturais, que dão ênfase a estrutura do sistema, e os comportamentais, que mostram a dinâmica do *software*. Para auxiliar o desenvolvimento da modelagem do sistema são comumente utilizados os diagramas *Unified Modeling Language* (UML) (FOWLER, 2004).

Na Figura D.2 é mostrado o diagrama de classes, dando atenção às principais classes que constituem o sistema, como também a interface de comunicação entre elas. Nessa Figura, é possível observar que o sistema desenvolvido utilizou três pacotes, um chamado *model*, outro chamado *control* e o último chamado *view*. O *model* é responsável pela lógica e as regras do sistema, como também a comunicação com o R para a predição e geração dos gráficos, de forma que o *model* envia os dados para o R, e após o R fazer os cálculos das predições, ele escreve todos os gráficos como imagens na pasta */tmp* do Linux, e o *model* é o responsável por buscar as informações dessas imagens e enviar para o *control*. Já o pacote *control* é resposável por fazer a comunicação entre o pacote *model* e *view*. Nesse projeto também foi usado o padrão FACADE, o qual prover uma interface simples do pacote *model* para o pacote *view*. Por fim, a *view* é onde o usuário entra com as informações e visualiza os gráficos com as predições.

Figura D.2: Diagrama de Classes

D.2 Funcionalidades

Ao iniciar a execução do ferramenta Predictor, o usuário encontrará a tela inicial do sistema mostrada na Figura D.3. Essa tela solicita do usuário que entre com a base de dados, sendo o primeiro passo para iniciar o processo de predição.

Há na Figura D.3 os botões *Help*, *Back* e *Next*, que estarão em todas as telas do sistema, e o botão *Load Database*, que serve para carregar a base de dados no sistema. Após carregar, a base de dados é imediatamente visualizada na tabela que se encontra nessa tela. A Figura D.4 demonstra isso.

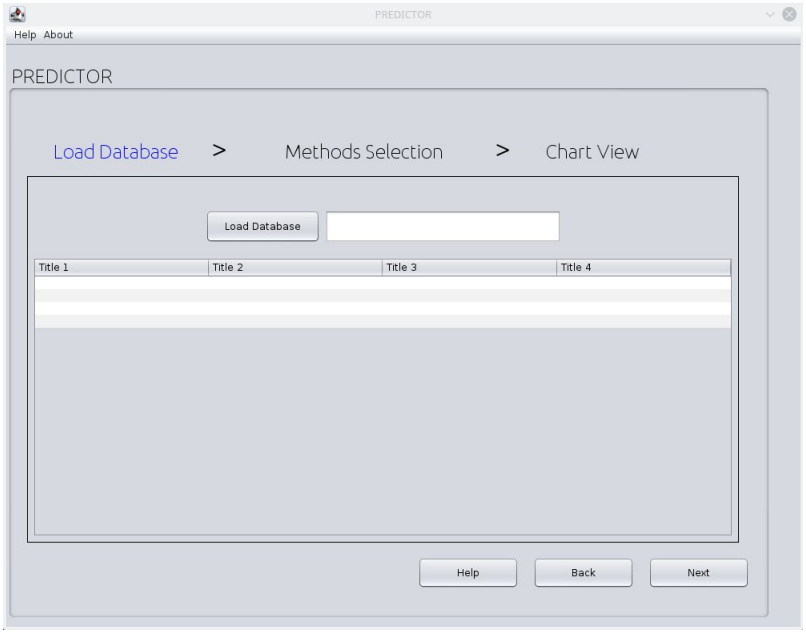


Figura D.3: Tela inicial da ferramenta Predictor

Essa tabela não está disponível para edição do usuário a partir da ferramenta para que evite uso indevido em algum campo e faça uma predição errônea. A base de dados é mostrada na tabela apenas para que o usuário possa conferir o nome das colunas e os dados para que tenha certeza que essa é a base que o mesmo deseja fazer a predição.

Após carregar a base de dados, o usuário clica no botão *Next*, então a tela com as opções das técnicas de predição aparece. O usuário pode selecionar uma ou mais técnicas, inclusive todas, caso ele não tenha conhecimento de como a série temporal se comporta. Desse modo, o usuário pode usar todas as técnicas e deixar a cargo do Predictor mostrar qual é a melhor.

A Figura D.5 mostra a tela com as técnicas de predição. Ao lado das opções há um pequeno campo ao lado direito com a descrição geral de cada técnica, a mesma é ativada mostrando as descrições quando o usuário passa o *mouse* em cima da técnica.

Depois de escolher as técnicas que serão usadas, o usuário clica novamente em *Next* e então a tela com as configurações do gráfico e da imagem aparece (Figura D.6). Essas configurações também afetam como a predição será feita, porém apenas três campos são obrigatórios, os

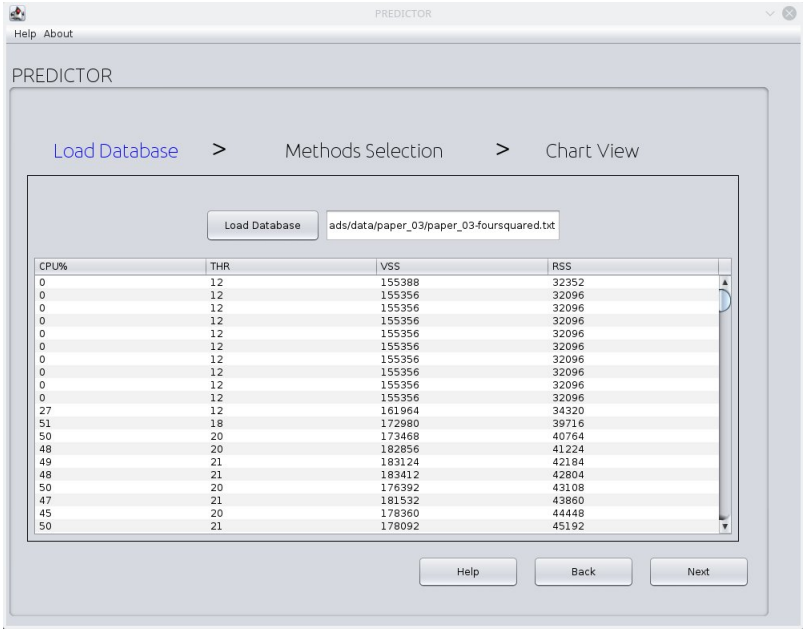


Figura D.4: Após carregar a base de dados.

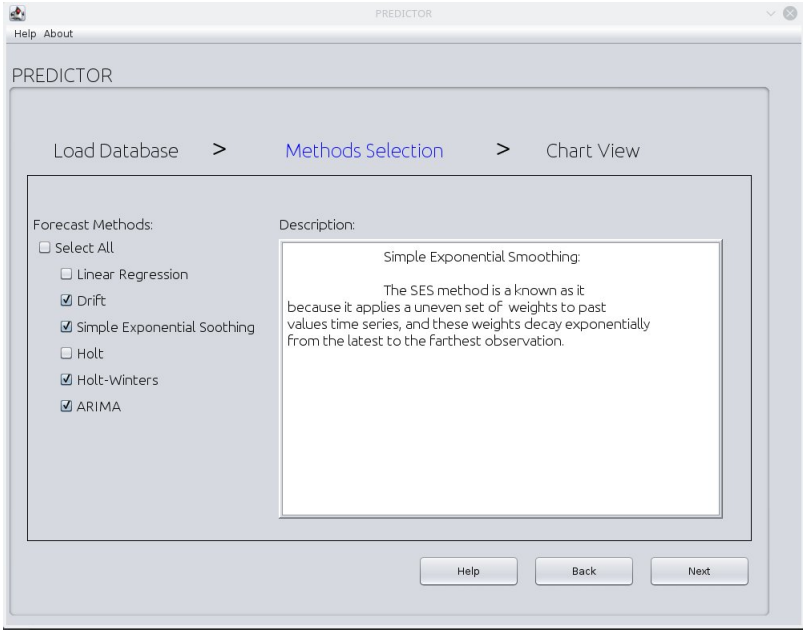


Figura D.5: Técnicas de predição.

outros ficam como configurações padrão, caso o usuário os deixe em branco.

A Figura D.6 mostra a tela das configurações. Nota-se que essa tela é divida em duas categorias, *Chart* e *Image*. A categoria *Chart* tem as configurações relacionadas ao gráfico que será gerado e a predição em si. Já a categoria *Image* é relacionada à imagem, onde o usuário pode definir a dimensão da mesma e se quer que as imagens sejam geradas em uma mesma janela ou em janelas diferentes. Caso o usuário opte pela visualização em janelas separadas, o Predictor fornece a opção para salvar o gráfico como arquivo de extensão PNG.

Depois de configurar o gráfico, o usuário clica no botão *Finish* e então os gráficos são

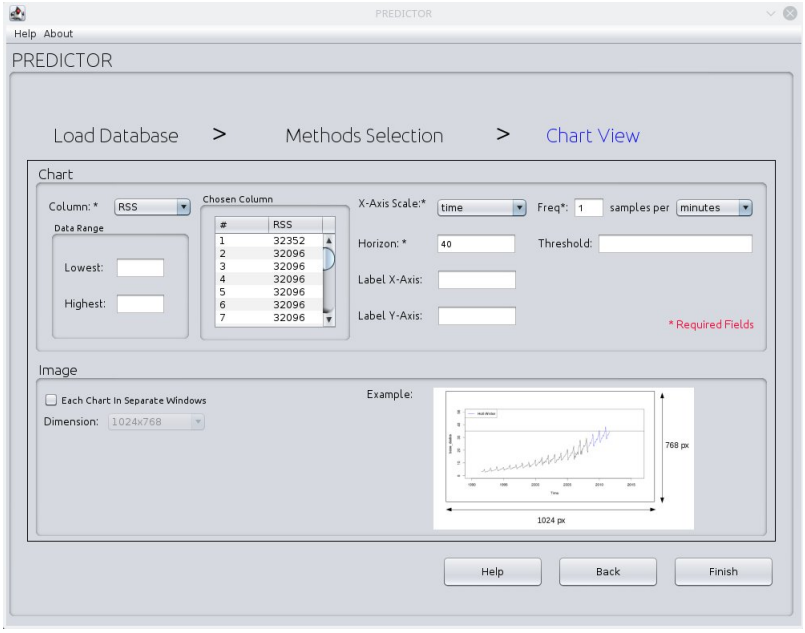


Figura D.6: Configurações da predição.

apresentados em outras janelas. A Figura D.7 mostra os gráficos que foram criados todos em uma única janela e a Figura D.8 mostra os gráficos gerados em janelas diferentes. Na Figura D.7 nota-se nos gráficos menores que o treinamento com a predição e os dados reais em vermelho. Assim o usuário pode ver qual das técnicas comporta-se como os dados reais. O conjunto de treinamento como dito anteriormente possui 80% dos dados da base, os 20% restantes formam o conjunto de teste. O conjunto de teste é usado para comparar com o conjunto de predição. Todas as técnicas selecionadas pelo usuário tem os gráficos gerados com esses conjuntos de dados, e somente o que tiver maior acurácia é executado novamente com a base de dados inteira.

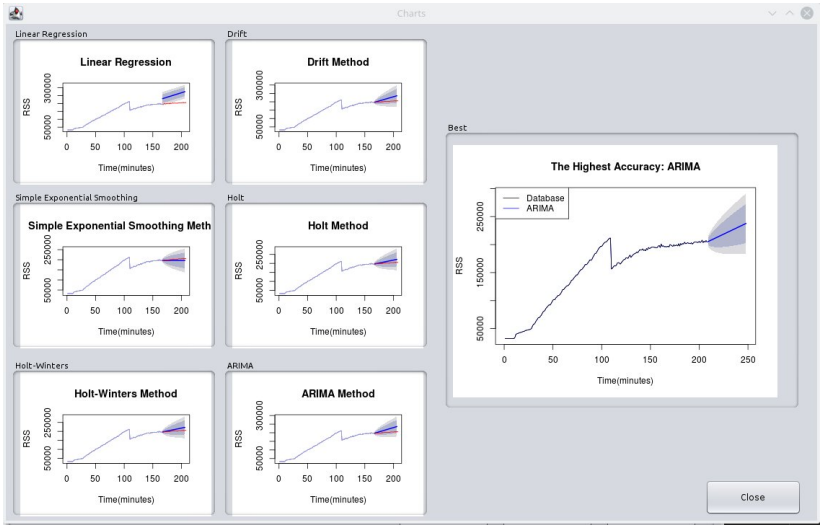


Figura D.7: Gráficos em uma janela.

A Figura D.8 mostra a técnica com maior acurácia e executada com a base de dados completa. A sombra que envolve a predição é o intervalo de confiança. Na Figura D.8 o campo

Threshold não foi preenchido.

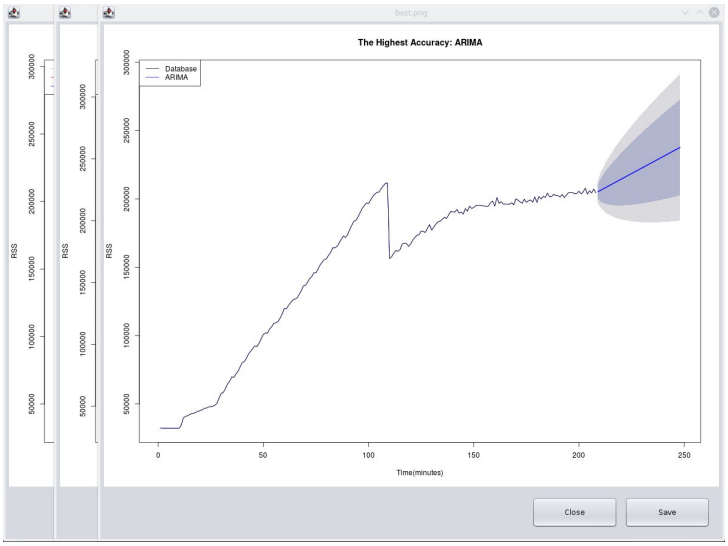


Figura D.8: Gráficos em diferentes janelas.

A Figura D.9 mostra como o gráfico é gerado quando o campo *Threshold* é preenchido. Esse campo cria um limiar no gráfico e quando esse limiar é atingido, uma linha vertical é inserida no gráfico para mostrar em qual lugar exatamente essa interseção ocorreu. Com essa informação o administrador do sistema pode tomar decisões de quando aplicar ações de prevenção à falhas de forma que a disponibilidade esteja otimizada o máximo possível.

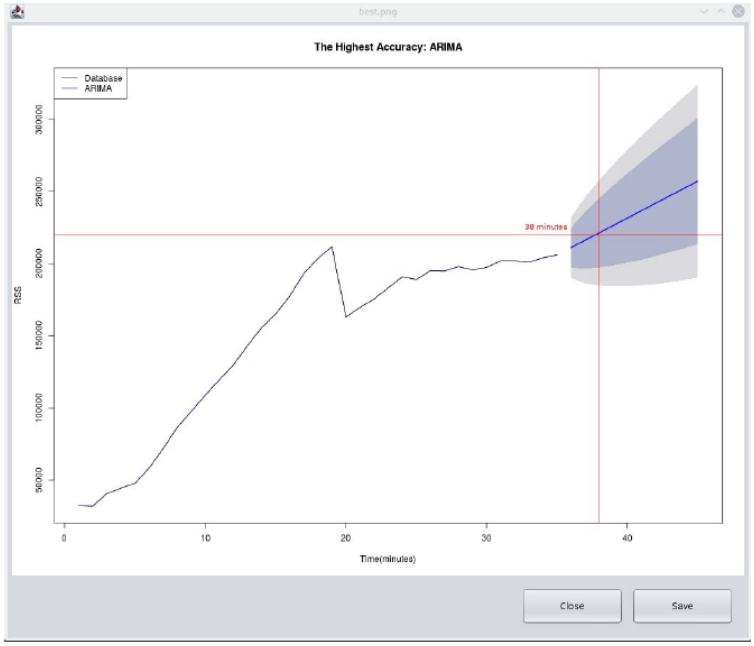


Figura D.9: Gráfico gerado com *threshold* preenchido pelo usuário.