DISSERTAÇÃO DE MESTRADO

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM

CIÊNCIA DA COMPUTAÇÃO

"Um Modelo Auto-Adaptativo para Otimização do Offloading em Computação Móvel nas Nuvens"

ALUNO: Flávio Akira Nakahara
ORIENTADOR: Prof. Dr. Delano Medeiros Beder

São Carlos, SP
Fevereiro/2018

CAIXA POSTAL 676
FONE/FAX: (16) 3351-8233
13565-905 - SÃO CARLOS - SP
BRASIL

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MODELO AUTO-ADAPTATIVO PARA APOIO AO OFFLOADING DINÂMICO EM APLICAÇÕES MÓVEIS

FLÁVIO AKIRA NAKAHARA

ORIENTADOR: PROF. DR. DELANO MEDEIROS BEDER

São Carlos – SP

Fevereiro/2018

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MODELO AUTO-ADAPTATIVO PARA APOIO AO OFFLOADING DINÂMICO EM APLICAÇÕES MÓVEIS

FLÁVIO AKIRA NAKAHARA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software

Orientador: Prof. Dr. Delano Medeiros Beder

São Carlos – SP

Fevereiro/2018

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de Dissertação de Mestrado do(a) candidato(a) **Flávio Akira Nakahara**, realizada em **20 de fevereiro de 2018.**

Prof. Dr. Delano Medeiros Beder
(UFSCar)

Prof. Dr. Jó Ueyama
(USP)

****

Prof. Dr. Danielo G. Gomes
(UFC)

Certifico que a sessão de defesa foi realizada com a participação à distância do membro Danielo G. Gomes, depois das arguições e deliberações realizadas, o participante à distância está de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa do(a) aluno(a).

Prof. Dr. Delano Medeiros Beder
Presidente da Comissão Examinadora
(UFSCar)

Dedicado à estimadíssima namorada Regina

AGRADECIMENTOS

Primeiramente agradeço ao meu orientador Prof. Dr. Delano Medeiros Beder pela orientação na realização deste trabalho, discussões de ideias, cobranças, e também conversas tanto acadêmicas como pessoais. Com certeza uma das pessoas mais rigorosas para a escrita, contribuindo enormemente para sua elaboração.

Agradeço também à minha mãe Dilce Akemi Nakahara, por dar todo o suporte tanto antes da pós-graduação, durante a escolha por realizá-la, e também durante sua realização. À minha tia Harumi Nakahara e a todos os familiares por todo o apoio oferecido.

Agradeço à minha namorada e companheira Regina Rocha Mattazio, que tanto me acompanhou durante esta jornada, seja através de suporte emocional, suporte técnico durante os experimentos, estando presente durante todos os eventos importantes sejam acadêmicos ou não, e aconselhando. Sem você este trabalho teria sido muito mais difícil.

Aos amigos Ana Braatz, Débora Hiroki, Hério Souza, Camila Kamimura e Glesio de Paiva da computação da UFSCar pelas longas horas de estudo, muitas conversas e diversão nas horas vagas. Aos companheiros de laboratório Felipe Giuntini e Fernanda Zampieri por tanto ajudarem nos apertos do mestrado como partilharem das comemorações.

Agradeço à UFSCar, por oferecer esta oportunidade e todas as condições para tornar esta experiência possível.

Ao Prof. Dr. Danielo Gomes pelas contribuições a este trabalho e participação das bancas de qualificação e defesa de dissertação. Também agradeço ao Phillip Costa e demais pesquisadores do grupo de pesquisas GREat da Universidade Federal do Ceará, responsáveis pelo trabalho utilizado como base para a realização deste, e também pela prontidão para responder minhas dúvidas.

Ao Prof. Dr. Jó Ueyama por ter aceito participar desta banca e contribuir para este trabalho.

Ao Prof. Dr. Daniel Lucrédio por ter participado da banca de qualificação e contribuído para a realização deste trabalho.

À CAPES, pelo apoio financeiro oferecido durante este mestrado.

Por fim dedico esta dissertação para aqueles que ainda lutam em busca de encontrar aquilo que querem realizar, e também para aqueles que encontraram seu caminho.

RESUMO

Computação móvel nas nuvens é um dos principais meios de melhorar desempenho de dispositivos móveis com recursos restritos, trazendo recursos e serviços de servidores remotos computacionalmente mais poderosos para prover suporte à execução de aplicações móveis mais robustas. Entretanto, o uso inteligente e eficiente dos recursos das nuvens é necessário devido às condições ambientais inconstantes e variabilidade de uso das aplicações. Esta dissertação apresenta CoSMOS - *Context-Sensitive Model for Offloading System* - um modelo de tomada de decisão ao *offloading* ciente de contexto e autoadaptativo para computação móvel nas nuvens, baseado em padrões de arquitetura para sistemas autoconscientes e autoexpressivos. Ele utiliza estimativas de tomada de decisão baseada em tempo de execução e consumo energético da aplicação para decidir eficientemente quando e quais métodos da aplicação serão migrados, com a finalidade de melhorar execução do sistema. Dois estudos de caso prático foram utilizados para avaliar a performance da abordagem do modelo: uma aplicação do problema das N rainhas, e BenchImage, do MpOS. Os resultados apresentados indicam que o modelo é capaz de inferir decisões apropriadas com performance aceitável em uma gama de condições de ambiente.

Palavras-chave: Computação móvel nas nuvens. Suporte à tomada de decisão. Ciência de contexto.

ABSTRACT

Mobile cloud computing is one of the main ways to augment the performance of resource-constrained mobile devices, bringing resources and services from computationally powerful remote servers in order to provide support to the execution of rich mobile applications. However, an efficient and intelligent use of cloud resources is required due to changing environment conditions and application variability usage. This dissertation presents CoSMOS - Context-Sensitive Model for Offloading System - a context-aware and self-adaptive offloading decision support model for mobile cloud computing systems, based on self-aware and self-expressive system architecture patterns. It employs decision-taking estimation based on application's time execution and energy consumption to decide efficiently when and which application methods should be offloaded in order to improve system's execution. Two practical study cases were used to evaluate the model's approach performance: a N-queen problem application, and MpOS's BenchImage. The results shown that the model is capable of inferring appropriate decisions with acceptable performance in a range of environment conditions.

Keywords: Mobile cloud computing. Decision taking support. Context awareness.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

GLOSSÁRIO

API – *Application Program Interface - Interface de Programacão de Aplicacões*

IDE – *Integrated Development Environment - Ambiente de Desenvolvimento Integrado*

MAPE-K – *Monitorar, Analisar, Planejar e Executar sobre um conhecimento (Knowledge) compartilhado*

MCC – *Mobile Cloud Computing - Computacão Móvel nas Nuvens*

MVC – *Model-View-Controller - Modelo-Visão-Controlador*

REE – *Remote Environment Execution - Ambiente de Execucão Remota*

RTT – *Round-Trip Time - Tempo de Ida e Volta*

SAS – *Self-Aware System - Sistema Autoconsciente*

UML – *Unified Modelling Language - Linguagem de Modelagem Unificada*

Capítulo 1

INTRODUÇÃO

Este capítulo apresenta o contexto e a motivação desta pesquisa, os problemas a serem abordados, os objetivos propostos, além da organização do trabalho.

1.1 Contexto da Pesquisa

Nos últimos anos, com a popularização dos *smartphones*, pessoas ao redor do mundo passaram a incorporar mais os dispositivos móveis em sua rotina, utilizando-os em frequência muito maior do que os dispositivos *desktop* (COMSCORE, 2014). O crescimento no uso de *smartphones* e demais dispositivos móveis é acompanhado pelo crescimento e expansão das aplicações móveis, que trazem diferentes formas de experiência de uso para diversos segmentos de uso, tais como entretenimento, saúde, jogos digitais, negócios, redes sociais, turismo e notícias (FERNANDO; LOKE; RAHAYU, 2013).

Entretanto essa crescente expansão na gama de formas de utilização de aplicações móveis implica em uso e desenvolvimento de ferramentas e aplicações cada vez mais complexas, que acabam por comprometer duas limitações críticas dos dispositivos móveis: (i) vasta diversidade de configurações de *hardware*, e (ii) capacidade energética limitada.

Enquanto a diversidade de configurações compromete na experiência de uso ao oferecer diversos níveis de imersão para uma mesma aplicação, a limitação na capacidade energética compromete o tempo útil de utilização das aplicações. Mesmo com a constante evolução das capacidades dos dispositivos móveis, essa evolução ainda não é suficiente para acompanhar a crescente demanda de recursos por parte das aplicações móveis (KUMAR et al., 2013; CHEN et al., 2017).

Uma das soluções mais comumente propostas para contornar tais limitações é por meio do uso da computação móvel nas nuvens[1], que objetiva trazer recursos computacionais de servidores para auxiliar na execução de funcionalidades de aplicações móveis com alto consumo de recursos. Por meio deste apoio busca-se diminuir a carga computacional e o armazenamento de dados dentro dos dispositivos móveis, implicando em melhoria na performance de processamento e na economia de energia (BAHTOVSKI; GUSEV, 2014).

1.2 Motivação e Problema de Pesquisa

Apesar de existirem pesquisas consolidadas na área de MCC e em *offloading* computacional de aplicações móveis para servidores remotos, os benefícios de seu uso na prática mostram-se instáveis, sendo capaz de até deteriorar o desempenho da execução pelo dispositivo móvel com suporte (KUMAR et al., 2013; AHMED et al., 2017; ZHOU; BUYYA, 2018). Alguns dos fatores principais que impactam na instabilidade da eficiência do suporte e limitando severamente seu uso são as especificidades técnicas de cada aplicação e as condições instáveis da rede de comunicação entre dispositivo e servidor remoto (AHMED et al., 2017; ZHOU; BUYYA, 2018).

Devido a essas limitações do suporte nas nuvens é necessário que a execução de aplicações seja otimizada com o uso de *design* eficiente da aplicação, desenvolvimento eficiente (AHMED et al., 2015), e otimização de uso dos recursos das nuvens de acordo com contexto tanto da aplicação como das condições de rede (KHAN et al., 2015; PARANJOTHI; KHAN; NIJIM, 2017).

Para atender os requisitos de diversos tipos de aplicações móveis diversos pesquisadores propuseram *frameworks* de plataformas para apoio de execução a aplicações móveis, que possuem como principal objetivo as otimizações de tempo de execução e de uso de recursos oferecidos pelo próprio dispositivo móvel. Este apoio é realizado através da migração da aplicação completa, ou de parte de seus componentes, para servidores remotos nas nuvens para que sejam executadas remotamente em servidores nas nuvens com recursos mais abundantes. Ao término da execução externa seus resultados retornariam para a aplicação móvel original, assim dando continuidade ao fluxo original de forma transparente ao usuário. Por meio do uso da plataforma de apoio busca-se atingir execução mais fluída possível e melhorar usabilidade com o mínimo de intervenção do usuário (AHMED et al., 2017).

[1] Em inglês: Mobile Cloud Computing (MCC).

1.3 Objetivos do Trabalho

O objetivo deste trabalho é a proposição de modelo de otimização de uso da técnica de *offloading* computacional em computação móvel nas nuvens para aplicações móveis que (i) seja ciente de contexto de execução onde a aplicação encontrar-se inserida; (ii) avalie mais de um parâmetro de otimização simultaneamente; (iii) adote como base estrutural uma arquitetura para sistemas autoadaptativos. Este modelo proposto é denominado CoSMOS (*Context-Sensitive Model for Offloading System* - Modelo Sensível ao Contexto para Sistemas de *Offloading*).

Através do modelo CoSMOS busca-se adaptar eficientemente aplicações móveis para permitir uso de plataformas de apoio por MCC, desta forma otimizando a execução de cada aplicação de acordo com as condições do ambiente de execução dentro do dispositivo móvel, com este apoio sendo realizado de forma transparente para o usuário.

Para desenvolver o modelo foram adotados padrões de arquitetura para sistemas autoconscientes e auto-expressivos (LEWIS et al., 2011), e como base de desenvolvimento e aplicação o *framework* MpOS (COSTA et al., 2015) e, portanto, foi adotada sua estrutura de suporte à migração: granularidade de migração em nível de método e suporte a execução remota em *cloudlets* e servidores remotos.

Como objetivos específicos foram adotados: (i) desenvolvimento do modelo de tomada de decisão a partir dos padrões de arquitetura para sistemas autoconscientes e auto-expressivos, (ii) avaliar inferência de comportamento em parâmetros por funções matemáticas de estimação através de estudo prático, e (iii) comparação de desempenho com *framework* mono-objetivo.

1.4 Metodologia do Trabalho

Para a realização deste trabalho foram empregadas quatro etapas, que se encontram brevemente descritas a seguir:

1. **Revisão da Literatura:** Foram realizadas revisões da literatura sobre os seguintes temas: arquitetura de sistemas autoadaptativos, e computação móvel nas nuvens. Dentro de suas respectivas áreas foram coletados diversos artigos e livros que buscaram focar e analisar aspectos isolados dentro da área, *surveys* gerais sobre o tema, e *frameworks* existentes de suporte à execução de aplicações;

2. **Análise e Definição da Arquitetura do Modelo:** Foram realizados levantamento de requisitos necessários para a elaboração do modelo. Com esta base foram analisados

os padrões de projeto para sistemas auto-adaptativos existentes, assim selecionando os mais adequados para elaboração do modelo de otimização de *offloading* por múltiplos parâmetros;

3. **Desenvolvimento do Modelo:** Com base na arquitetura proposta na fase anterior, foi elaborado o modelo para otimização de tomada de decisão e suas operações, além de sua integração a um *framework* já existente;

4. **Avaliação de Resultados:** Após o desenvolvimento do modelo, foram utilizadas duas aplicações como meio de avaliação de desempenho, sendo todos de alta carga de processamento computacional. Todas as avaliações foram realizadas sob seis condições distintas de execução.

1.5 Organização da Dissertação

O presente trabalho encontra-se disposto em 6 capítulos, incluindo este, com os próximos organizados da seguinte forma:

Capítulo 2: O capítulo apresenta os principais conceitos relacionados a este trabalho sobre sistemas autoadaptativos e computação móvel nas nuvens. Ao final são apresentados trabalhos relacionados e um estudo comparativo entre esses e o trabalho proposto CoSMOS;

Capítulo 3: O capítulo apresenta a proposição do CoSMOS através da elicitação dos requisitos funcionais e não-funcionais e restrições do modelo, análise e seleção dos padrões de projeto para sistemas autoconscientes e auto-expressivos, e escolha da forma como é realizada a representação dos parâmetros dentro do modelo;

Capítulo 4: O capítulo apresenta o fluxo de operação do CoSMOS e as fases de tomada de decisão de *offloading*, suas respectivas abordagens, estruturas e desenvolvimento;

Capítulo 5: O capítulo apresenta as duas aplicações móveis utilizadas para a avaliação e validação do CoSMOS. Também neste capítulo é discutido os resultados obtidos durante a execução dos experimentos;

Capítulo 6: O capítulo descreve de forma resumida os resultados alcançados, bem como as conclusões obtidas a partir deste trabalho. Também são apresentadas possíveis melhorias a serem consideradas em trabalhos futuros.

Capítulo 2

REVISÃO BIBLIOGRÁFICA

Este capítulo tem como objetivo discutir conceitos, abordagens e técnicas no âmbito de arquiteturas para sistemas autoadaptativos e computação móvel nas nuvens, juntamente com comparações entre alguns dos *frameworks* de computação móvel nas nuvens mais relevantes na área, com a finalidade de prover base teórica acerca dos aspectos gerais relacionados a este trabalho.

2.1 Arquitetura para Sistemas Autoadaptativos

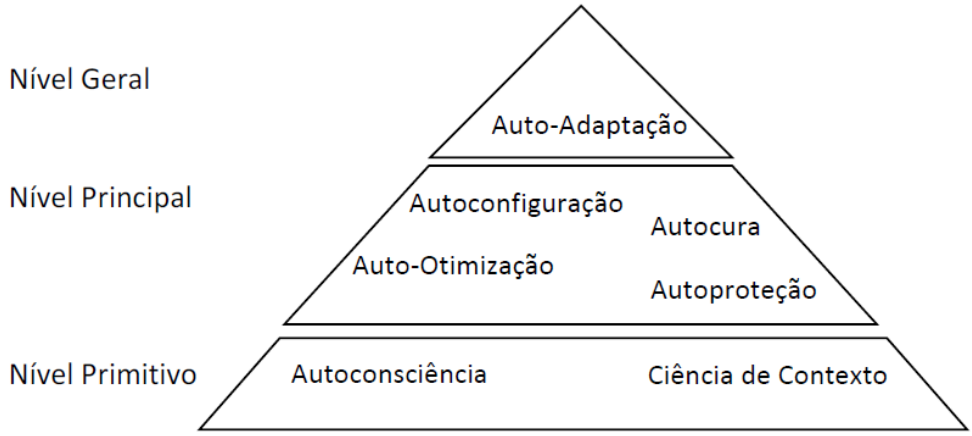
Um sistema auto-adaptativo pode ser definido como um sistema ou conjunto de sistemas que podem adaptar suas configurações durante a execução conforme as adversidades encontradas em prol de um objetivo a eles determinados, sem a necessidade de intervenção humana (KEPHART; CHESS, 2003).

Segundo Kephart e Chess (2003), o uso de sistemas auto-adaptativos torna-se necessário para manter o funcionamento de sistemas que se tornam cada vez mais complexos e massivos, extrapolando até mesmo a capacidade de humanos habilidosos em mantê-los operacionais e de tomar decisões rápidas e precisas.

2.1.1 Propriedades Auto-*

Pesquisadores da IBM (IBM, 2005) liberaram um manifesto no qual encontram-se compiladas sete propriedades, conhecidas como ”propriedades auto-*”. Salehie e Tahvildari (2009) apresentaram essas propriedades dispostas em hierarquia e dividindo-as em três níveis: nível geral, nível principal e nível primitivo. A Figura 2.1 ilustra este modelo proposto.

Figura 2.1: Hierarquia de Propriedades Auto-*



Adaptado de Salehie e Tahvildari (2009).

- **Nível Geral:** abrange as propriedades globais de sistemas autoadaptativos. Um subconjunto de propriedades dentro de autoadaptação abrange autogovernança, autogerenciamento, autocontrole e auto-organização (SALEHIE; TAHVILDARI, 2009).

- **Nível Principal:** compreendem quatro propriedades inerentes à computação autoadaptativa, definidas com base em mecanismos biológicos de autoadaptação (KEPHART; CHESS, 2003).

  – Autoconfiguração: relaciona-se à capacidade do sistema de configurar a si mesmo de acordo com os objetivos de alto nível estabelecidos para si, sendo capaz de instalar novos programas e preparar a si próprio de acordo com as necessidades.

  – Auto-otimização: relaciona-se à capacidade do sistema otimizar seus próprios recursos para melhor execução, seja reativamente ou proativamente.

  – Autocura: relaciona-se à capacidade do sistema de detectar e diagnosticar problemas presentes em si, desde erros em dados até falhas graves, e caso possível repará-los.

  – Autoproteção: relaciona-se à capacidade do sistema se proteger tanto de ataques externos maliciosos como contra usos indevidos, antecipando possíveis brechas de segurança e prevenindo-as de ocorrer futuramente.

- **Nível Primitivo:** abrangem as propriedades diretamente ligadas à compreensão do sistema sobre o ambiente interno e externo.

  – Autoconsciência: segundo Hinchey e Sterritt (2006), significa que o sistema está ciente de seu estado de execução bem como seu comportamento. Refere-se principalmente a mudanças internas, tais como mudanças de estado ou falhas que occoram durante sua execução.

  – Ciência de contexto: significa que o sistema está ciente do ambiente operacional no qual encontra-se inserido, referindo-se principalmente a eventos externos que influencie em mudanças, tais como requisições de usuários.

Segundo Salehie e Tahvildari (2009), acredita-se que as propriedades de auto-* estejam fortemente relacionadas a fatores de qualidade de software, auxiliando na compreensão de fatores, métricas e requisitos de qualidade para desenvolvimento de software.

No nível principal, as quatro propriedades possuem grande influência nos seguintes fatores de qualidade:

- **Autoconfiguração:** Manutenibilidade, funcionalidade, portabilidade e usabilidade;
- **Autocura:** disponibilidade, capacidade de sobrevivência, manutenibilidade e confiabilidade;
- **Auto-otimização:** eficiência, também impactando em funcionalidade;
- **Autoproteção:** forte ligação com confiabilidade, e também com funcionalidade.

Já o nível primitivo possui forte ligação com fatores de manutenibilidade, funcionalidade e portabilidade.

2.1.2 Sistemas Autoconscientes e Auto-Expressivos

Baseando-se nos trabalhos no campo da psicologia sobre autoconsciência, Lewis et al. (2011) desenvolveram categorias de autoconsciência a partir de trabalhos já existentes sobre autoconsciência em sistemas computacionais, organizando-os para uso em sistemas autoadaptativos.

E a partir dessa base estabelecida com as categorias anteriormente propostas, Chen et al. (2014) formularam sete padrões de desenvolvimento de sistemas autoconscientes, cada qual com funções e finalidades específicas. Padrões arquiteturais e metodologia para o desenvolvimento de sistemas autoconscientes e autoexpressivos também foram desenvolvidos e aplicados em casos de estudo (DUTT; JANTSCH; SARMA, 2016).

Enquanto a autoconsciência se preocupa com o conhecimento interno do próprio sistema e de cada componente, a capacidade de determinar os efeitos advindos de cada decisão tomada e seu impacto dentro e fora do sistema é nomeada capacidade de auto-expressão (LEWIS et al., 2011).

Inspirando-se na teoria de Duval e Wicklund (1972) sobre a divisão da autoconsciência entre autoconsciência objetiva – que trata sobre a introspecção da própria pessoa sobre si mesmo – e a autoconsciência subjetiva – que trata sobre as ações e experiências advindas da própria pessoa –, a autoconsciência computacional pode ser similarmente dividida em dois tipos de autoconsciência:

> **Autoconsciência pública:** reúne conhecimento sobre o ambiente no qual o sistema encontra-se inserido, seja advindo de outros sistemas ou de sensores externos a si.
>
> **Autoconsciência privada:** reúne conhecimento sobre o próprio sistema através de sensores internos a si. Tal conhecimento não se encontra normalmente acessível por sistemas externos, salvo quando disponibilizado.

Para que um sistema possa ser considerado totalmente autoconsciente tanto no sentido privado como público, este deve ser capaz de adquirir conhecimento acerca do ambiente externo e de seu próprio estado interno de execução (LEWIS et al., 2011).

Além da divisão de autoconsciências pública e privada, inspirando-se no modelo de cinco níveis de autoconsciência humana proposto pelo psicólogo Ulric Neisser (1997), foram propostos cinco níveis de planejamento de ação em sistemas autoconscientes, cada um destes utilizando abordagens únicas de planejamento com base em conjuntos distintos de informações dentre autoconsciência pública, privada, histórico de execução e linhas-guia do próprio sistema. A relação dos níveis de autoconsciência, suas funções dentro do sistema e o nível equivalente no modelo psicológico são apresentados na Tabela 2.1.

Tabela 2.1: Comparação entre o modelo do *framework* computacional de autoconsciência e o modelo psicológico proposto por Neisser

Nível no modelo de SAS[1]	**Definição no modelo de SAS[1]**	**Nível equivalente no modelo de Neisser**	**Definição por Neisser**
Consciência de estímulo	O sistema possui consciência sobre os estímulos provindos de sensores externos, e pode responder adequadamente a eles.	Eu ecológico	Como a pessoa percebe o ambiente ao seu redor.
Consciência de interação	O sistema, com consciência dos estímulos externos e de seu atual estado, pode elaborar ações de interação com outros sistemas e com o ambiente.	Eu interpessoal	Como a pessoa interaje com o ambiente e outros seres.
Consciência de tempo	O sistema possui conhecimento sobre ações passadas, assim permitindo planejar ações futuras.	Eu estendido	Baseado principalmente nas memórias e capacidade de antecipação da pessoa.
Consciência de objetivo	O sistema possui conhecimento sobre seu objetivo principal, prioridades a serem tomadas durante o processamento de dados, bem como suas restrições.	Eu privado	Trata sobre experiências pessoais e sentimentos da pessoa.
Meta autoconsciência	O sistema possui conhecimento sobre si mesmo, seus próprios níveis de consciência, e pode adaptar cada uma conforme necessidade.	Eu conceitual	Como a pessoa atribui significância a si mesma e às próprias ações.

Adaptado de Lewis et al. (2015).

Após realizar processamento das informações através dos cinco níveis passa-se para o **nível de auto-expressão**, responsável por analisar o resultado e delegar para as devidas saídas de dados: interno ao próprio sistema, ou para o ambiente externo.

2.1.2.1 Padrões de Arquitetura

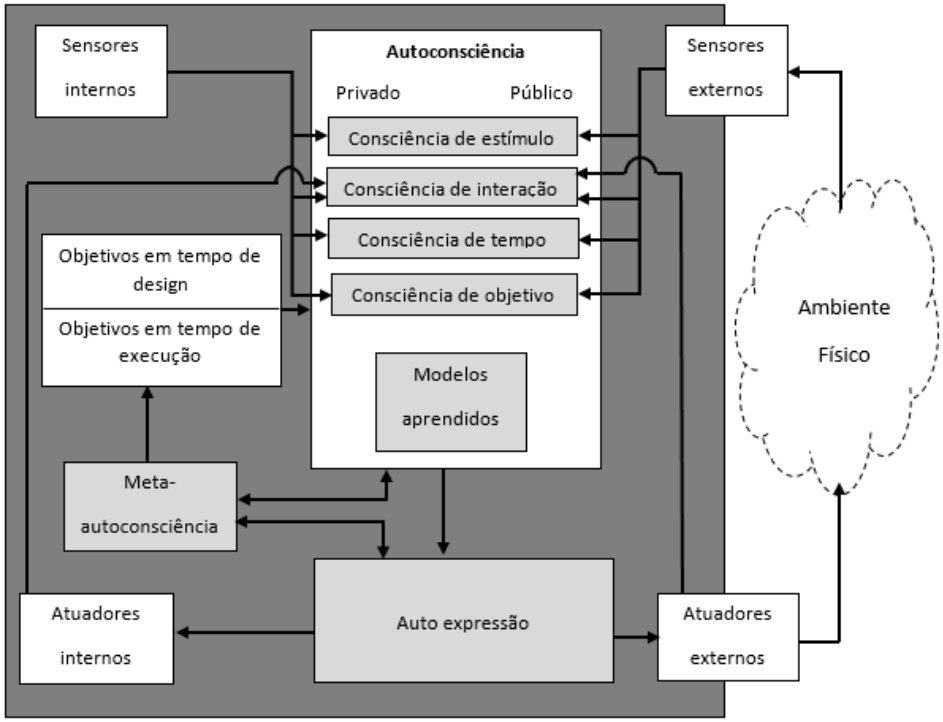
Tendo como base os cinco níveis de autoconsciência foram formulados sete padrões de arquitetura para sistemas autoconscientes. Cada padrão adota um determinado conjunto de níveis para atender as demandas necessárias.

- **Padrão Básico:** é o padrão de autoconsciência mais básico, composto somente pela consciência de estímulo;
- **Padrão Básico de Compartilhamento de Informações:** é composto pelas consciências de estímulo e interação para elaborar respostas mais complexas tanto aos outros sistemas como ao ambiente;
- **Padrão de Compartilhamento de Conhecimento Temporal:** para avaliar eventos passados e predizer novas ações este padrão utiliza consciências de estímulo, interação e tempo;
- **Padrão de Ciência de Conhecimento Temporal:** para respostas mais simples somente baseadas no histórico de execução, este padrão faz uso das consciências de estímulo e tempo;
- **Padrão de Compartilhamento de Objetivo:** para sistemas com mudanças constantes de objetivo principal, este padrão utiliza as consciências de estímulo, interação e objetivo;
- **Padrão de Ciência de Objetivo Temporal:** quando não há a necessidade de avaliar interação entre conjunto de sistemas, este padrão adota somente as consciências de estímulo, tempo e objetivo;
- **Padrão de Meta-Autoconsciência:** o padrão mais completo, utilizando consciências de estímulo, interação, tempo e objetivo em conjunto com meta-autoconsciência para gerenciar as decisões.

[1] Sistema autoconsciente, do inglês: Self-Aware System (SAS).

A Figura 2.2 ilustra o padrão de arquitetura de meta-autoconsciência para sistemas auto-conscientes e auto-expressivos.

Figura 2.2: Modelo de referência da arquitetura para sistema autoconsciente e auto-expressivo



Adaptado de Lewis et al. (2015).

2.1.2.2 Fluxo de Execução

Segundo Lewis et al. (2015), a manifestação da autoconsciência de um sistema se baseia no fluxo contínuo de dados provindos através de sensores, tanto àqueles internos ao sistema como aos externos a ele. Enquanto sensores internos podem monitorar temperatura e consumo de energia, sensores externos podem monitorar tanto o ambiente externo via microfone e câmeras como receber informações relativas a outros sistemas.

A partir de cada sensor os dados são enviados para todas as camadas de autoconsciência, onde cada uma será responsável por selecionar os dados relevantes, analisá-los e processar de acordo com seus critérios e funcionalidades. Após processados, esses dados serão encaminhados para a camada de autoexpressão, que decidirá a forma de saída das informações a serem externalizados através de atuadores, seja para ações internas ao próprio sistema como para ações no ambiente externo a ele.

Caso alguma informação necessite ajustes nos critérios adotados pelo sistema para manter seu funcionamento, essa informação será enviada para a camada de meta-autoconsciência, a qual será responsável por analisar, processar e encaminhar as novas instruções à camada de autoconsciência para que esta tome as devidas providências de acordo com as informações recebidas. As decisões sobre ajustes do sistema serão tomadas com base no histórico de eventos, custo-benefício de cada ajuste, *feedback* provindo do ambiente e estado no qual o sistema se encontra.

2.2 Computação Móvel nas Nuvens

Computação móvel das nuvens é uma tecnologia que busca trazer recursos computacionais provenientes de serviços das nuvens para fornecer suporte à execução de aplicativos em dispositivos móveis em diversas frentes (AHMED et al., 2015), sendo elas: complementar o poder de processamento, estender a capacidade de armazenamento de dados, redução no consumo energético e ampliar a segurança dos dados (SANAEI et al., 2012).

O suporte pelos serviços pode ser oferecido de diversas formas, como processamento paralelo de dados do aplicativo, armazenamento externo de dados, e prover informações externas ao dispositivo (FERNANDO; LOKE; RAHAYU, 2013).

Inúmeras aplicações para o uso de computação móvel nas nuvens surgiram nos últimos anos, entre elas comércio móvel, compartilhamento de mídias, aprendizado móvel, sensores móveis, aplicações móveis voltados à saúde, jogos móveis, mídias sociais móveis, serviços baseados em localização e realidade aumentada (WANG; CHEN; WANG, 2015).

2.2.1 *Offloading* Computacional

Offloading computacional, ou *cyber foraging*, é uma das técnicas mais comumente empregadas dentro de computação móvel nas nuvens. Esta técnica refere-se ao processo de migração de parte do processamento intenso de dados do aplicativo móvel para máquinas com maior capacidade de processamento computacional (PAMBORIS, 2014).

Dentro de MCC, esta técnica busca empregar para o processamento externo de alta intensidade o uso servidores com maior poder de processamento computacional localizados nas nuvens. Servidores que recebem processos provindos de dispositivos móveis localizados em suas proximidades são referidos como servidores *cloudlet* (BAHTOVSKI; GUSEV, 2014).

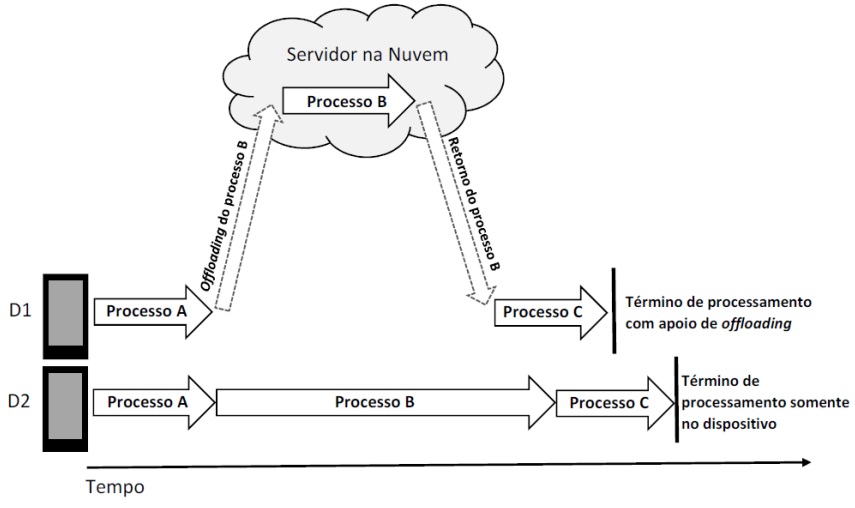
[1] Em inglês: Mobile Cloud Computing (MCC).

Segundo Zhou e Buyya (2018), existem quatro categorias de abordagem ao *offloading*:

- ***Offloading* Particionado:** busca migrar somente as partes da aplicação móvel com processamento mais intenso para os servidores remotos;
- **Migração para Máquina Virtual:** busca migrar imagens da aplicação móvel para servidores remotos que rodem máquinas virtuais dos dispositivos. Esta migração pode ser tanto de parte da aplicação como a aplicação completa;
- ***Offloading* Baseado em Agente Móvel:** Um agente intermediário realiza a comunicação entre dispositivo móvel e servidor remoto, buscando facilitar continuidade de execução das aplicações por parte do dispositivo e auxiliando em tolerância a falhas durante as execuções remotas;
- ***Offloading* Baseado em Replicação:** uma réplica idêntica da aplicação em execução no dispositivo móvel é mantida em execução no servidor remoto. Desta forma busca-se diminuir o volume de dados a serem trocados entre ambos em cada interação para um número maior de aplicações de diferentes naturezas.

Um exemplo que ilustra a diferença entre o uso ou não de *offloading* para auxiliar na execução de processos pode ser observado na Figura 2.3. Nesta figura encontram-sem ilustrados dois dispositivos móveis idênticos, identificados por D1 e D2, que executam três processos em série: processos A, B e C. Neste exemplo assume-se que ambos os dispositivos possuam especificações técnicas idênticas.

Figura 2.3: Comparação de tempo execução exclusivamente local e com auxílio de *offloading* de uma sequência de processos por dois dispositivos D1 e D2



Adaptado de Kim (2012).

Enquanto o dispositivo D2 executa todos os processos localmente, o dispositivo D1 utiliza *offloading* particionado para migrar somente o processo B, considerado como computacionalmente mais intenso, para ser executado em um servidor remoto na nuvem com maior poder computacional.

Pode-se observar que, apesar do acréscimo do tempo de tráfego de dados entre dispositivo e servidor no tempo total de execução dos três processos, estes foram executados em tempo menor com o auxílio do servidor no dispositivo D1 do que aqueles que foram executados exclusivamente dentro do dispositivo D2.

A situação ilustrada na Figura 2.3 somente apresenta melhora de performance através da diminuição do tempo de execução geral dos processos. Pode-se aplicar outras métricas de comparação para avaliar ganho de performance, tais como consumo energético, consumo de recursos computacionais, entre outros.

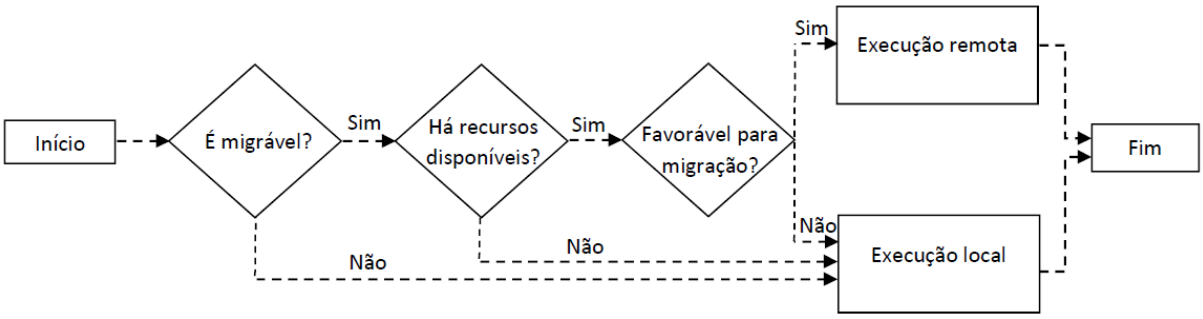
2.2.2 Execução de Aplicações Móveis em MCC

A execução de uma aplicação móvel que utiliza o apoio de MCC pode ser ilustrado como um diagrama de estados: normalmente encontra-se em estado inativo até que o usuário invoque sua chamada. A partir desse momento a aplicação passa do estado inativo para o estado de execução ativa, executando suas funcionalidades normalmente.

Ao encontrar uma funcionalidade candidata a *offloading*, o *framework* força a aplicação a entrar em estado de espera enquanto os processos referentes à migração são realizados.

Ao iniciar o processo de migração os seguintes passos são efetuados, conforme ilustrado na Figura 2.4.

Figura 2.4: Processo de *offloading* de funcionalidade



Adaptado de Khan et al. (2014).

Primeiramente é verificado se a funcionalidade é candidata ou não à migração. Em caso afirmativo verifica-se se há recursos disponíveis tais como presença de conexão com Internet e servidores acessíveis. Em caso afirmativo de disponibilidade de recursos, por fim, verifica-se as informações e condições acerca da funcionalidade para efetuar a migração, conforme estabelecido pelos objetivos do usuário e da aplicação.

Em caso da favorabilidade das condições for positiva, a funcionalidade e todos os dados relativos à sua execução serão enviadas para ser executada no servidor nas nuvens, com o resultado retornado diretamente para a aplicação, que dará prosseguimento à execução normal no estado ativo.

Caso algum dos passos tenha resultado negativo, a verificação será encerrada. Desta forma a execução da funcionalidade será realizada localmente, prosseguindo com o fluxo normal de execução da aplicação móvel.

2.2.3 Ciência de Contexto em Computação Móvel nas Nuvens

Apesar da computação móvel nas nuvens poder melhorar performance da execução de aplicações móveis através do uso de *offloading* computacional, seu uso não é sempre benéfico quando há fatores que o comprometem, tais como instabilidade na infraestrutura da rede de comunicação e restrições dos recursos. Então, ao invés de melhorar uso da aplicação, o *offloading* computacional pode causar degradação de performance ou desperdício energético (KHAN et al., 2015).

Dentro deste cenário, a habilidade de ciência de contexto é importante dentro da computação móvel nas nuvens, por permitir que sistemas possam alterar e adaptar suas configurações baseados em informações contextuais, com o objetivo de melhorar o ambiente de execução. Esta adaptação torna-se mais notável quando sujeito a ambientes instáveis e em constantes alterações (FERNANDO; LOKE; RAHAYU, 2013).

De acordo com Khan et al. (2015), existem dois principais modelos para a aplicação de ciência de contexto em computação móvel nas nuvens: (i) particionamento de aplicação e (ii) *offloading* computacional.

Particionamento de aplicação aborda o problema de divisão da aplicação em componentes de execução e distribuir entre o dispositivo móvel e servidor remoto na nuvem, pois o *offloading* completo da aplicação seria prejudicial à execução desta devido às dependências de recursos existentes exclusivamente no dispositivo e ao alto tráfego de dados e consumo energétivo pela migração. Este particionamento de aplicação pode ser tanto estático (estabelecido durante o

desenvolvimento da aplicação) como dinâmico (decidido durante sua execução, com base em condições do ambiente de execução).

Offloading computacional aborda a otimização do uso da técnica de migração de dados, pois esta pode levar a perda de desempenho para algumas aplicações por conta do consumo de recursos para tráfego de dados entre dispositivo móvel e servidor remoto. A instabilidade na infraestrutura de comunicação também afeta diretamente seu desempenho.

Este presente trabalho adotou como principal foco de estudo o modelo de *offloading* computacional, empregando técnicas de sistemas auto-adaptativos para otimização do uso de *offloading*.

2.2.4 *Frameworks* Existentes

Os *frameworks* de execução de aplicações móveis foram desenvolvidos para prover suporte a uma vasta gama de aplicações móveis com especificidades distintas e sendo executados em diversos ambientes. Seu principal objetivo é o de amplificar os recursos disponíveis às aplicações ao prover recursos e serviços de servidores que se encontram nas nuvens (AHMED et al., 2015).

E para este fim diversos *frameworks* foram desenvolvidos, cada qual possuindo abordagens distintas para diversas frentes, tais como infraestrutura de suporte, acoplamento às aplicações, particionamento da execução e tomada de decisão de *offloading* computacional.

Para efeito de classificação será adotado o modelo de divisão de *frameworks* proposto por Ahmed et al. (2015), separando de acordo com o número de objetivos que guiam a otimização de execução de aplicações móveis adotada por cada plataforma. Serão apresentados a seguir nove *frameworks* existentes na literatura, divididos em três categorias: otimização mono-objetiva, biobjetiva e multibjetiva.

2.2.4.1 *Frameworks* Baseados na Otimização Mono-Objetiva

Frameworks baseados na otimização mono-objetiva focam-se principalmente em otimizar somente um objetivo durante a execução. Por possuírem somente um foco, estes constituem a forma mais simples de otimização dentro do contexto de MCC, enfrentando dificuldades de performance em ambiente heterogêneo e de tornar execuções eficientes para algumas aplicações e condições de execução (AHMED et al., 2015).

A seguir serão apresentados dois dos mais relevantes *frameworks* da literatura baseados na otimização mono-objetiva: MAUI (CUERVO et al., 2010) e CloneCloud (CHUN et al., 2011).

MAUI: Cuervo et al. (2010) desenvolveram o *framework* MAUI, um sistema que pode ser integrado a aplicações móveis para permitir o *offloading* de funcionalidades em nível de método para um servidor remoto nas nuvens.

O foco de sua otimização é no custo energético de execução global da aplicação móvel. Para atingir este objetivo o sistema representa o fluxo de execução da aplicação móvel como grafo, onde cada vértice representa um método invocado durante a execução.

Para obter o particionamento dos métodos que provenha o melhor aproveitamento energético ao efetuar migração das partições obtidas, MAUI modela todos os vértices do grafo como problema matemático linear sujeito a restrições de infraestrutura de rede de comunicação, cuja solução para otimização do consumo energético é determinado através do uso de programação linear inteira.

CloneCloud: Chun et al. (2011) desenvolveram o *framework* CloneCloud, um sistema que realiza *offloading* das funcionalidades de aplicações móveis para máquina virtual localizada nas nuvens sem intervenção no código original, com o próprio *framework* responsável por particionar a execução da aplicação.

O foco de sua otimização é no custo geral de execução das aplicações. Para atingir este objetivo o *framework* particiona a aplicação móvel durante sua execução e realiza análise estática e dinâmica de cada parte, obtendo dados relativos ao custo de execução local, da execução remota, e custo de migração para servidor remoto. Esses dados são expressos como funções matemáticas, e a decisão de migração para cada partição é obtida ao minimizar todas as funções através do uso de programação linear inteira.

2.2.4.2 MpOS

Um *framework* que merece destaque devido à influência exercida sobre este trabalho é *Multiplatform Offloading System* (MpOS), desenvolvido por Costa et al. (2015).

Sua principal característica é a de oferecer suporte para *offloading* computacional em nível de método para mais de uma plataforma móvel simultaneamente, como *Android* e *Windows Phone*. Um mesmo servidor é capaz de oferecer apoio à execução remota de plataformas distintas através do uso de ambientes de execução remotos específicos, que são compostos por serviços de rede que processam operações de *offloading* para cada plataforma móvel.

O *framework* MpOS é formado por duas partes:

- MpOS API, incorporado junto ao código-fonte da aplicação móvel sobre a qual deseja-se aplicar o suporte, é responsável por marcar os métodos candidatos a *offloading*, enviá-los para o servidor remoto, receber os dados processados da chamada e prosseguir com o fluxo normal de execução;
- MpOS Plataforma, executado em um servidor remoto, é responsável por receber os dados da chamada de método da aplicação, executá-la e retornar o resultado do processamento para a aplicação original.

O principal foco de otimização do *framework* é de minimização de latência da rede entre dispositivo móvel e servidor nas nuvens. Para atingir este objetivo o *framework* monitora a latência da rede, efetuando a migração somente quando as condições de rede estiverem propícias para a migração.

2.2.4.3 *Frameworks* Baseados na Otimização Biobjetiva

Frameworks baseados na otimização biobjetiva focam-se em utilizar até dois objetivos simultaneamente para realizar a otimização de execução das aplicações e, portanto, possuem complexidade significantemente superior a *frameworks* baseados na otimização mono-objetiva.

Para efetuar a otimização biobjetiva estes sistemas incorporam dois parâmetros distintos em seu funcionamento, variando de *framework* para *framework*. Dada sua complexidade superior, estes *frameworks* são capazes de se adaptar melhor a ambientes de execução heterogêneos e de tornar execuções mais eficientes para uma gama maior de aplicações e condições de execução (AHMED et al., 2015).

A seguir serão apresentados dois *frameworks* baseados na otimização biobjetiva: AIOLOS (VERBELEN et al., 2012) e Cuckoo (KEMP et al., 2012).

AIOLOS: Verbelen et al. (2012) desenvolveram o *framework* AIOLOS, um sistema *middleware* que oferece apoio ao *offloading* adaptativo ao otimizar tanto a execução da aplicação como o servidor que efetuará a execução remota.

Os focos de sua otimização são no tempo de execução e latência de rede. Para atingir este objetivo o *framework* estima o tempo de execução local e remoto para cada funcionalidade através de funções matemáticas, geradas através de regressão linear com base no histórico de execução.

Cuckoo: Kemp et al. (2012) desenvolveram o *framework* Cuckoo, um sistema que oferece ferramentas de desenvolvimento de aplicações *Android* assim como plataforma para *offloading* de funcionalidades de aplicações móveis. Tais ferramentas são integradas diretamente ao ambiente de desenvolvimento, servindo como guia para o desenvolvimento das aplicações.

O foco de sua otimização é no custo de execução e economia de energia. Para atingir este objetivo o *framework* divide as funcionalidades entre componentes computacionalmente intensivas e componentes interativos através da linguagem de definição de interface do *Android*, uma linguagem desenvolvida para definir decomposição de objetos para uso em serviços de comunicação entre processos. Assim que os serviços referentes a cada funcionalidade encontram-se disponíveis no servidor, o *framework* decidirá quais serão migradas através de heurísticas simples, como diferença de tempo entre execuções local e remota, histórico de execução, e latência de rede.

2.2.4.4 *Frameworks* Baseados na Otimização Multiobjetiva

Frameworks baseados na otimização multiobjetiva focam-se na otimização de mais do que dois objetivos simultaneamente durante sua execução, portanto possuindo complexidade superior às categorias anteriormente citadas.

Para suas funções de otimização são incorporadas uma grande variedade de parâmetros distintos, tornando suas operações mais precisas e adequadas para execução de aplicações móveis. Dada sua complexidade, estes *frameworks* são ainda melhores para se adaptar a ambientes heterogêneos que os sistemas baseados em otimização biobjetiva, e também atendem a uma gama mais vasta de aplicações móveis e condições de execução (AHMED et al., 2015).

A seguir serão apresentados quatro *frameworks* baseados na otimização multiobjetiva: ThinkAir (KOSTA et al., 2012), MACS (KOVACHEV; YU; KLAMMA, 2012), MAsCOT (NAQVI et al., 2016) e Odessa (RA et al., 2011).

ThinkAir: Kosta et al. (2012) desenvolveram o *framework* ThinkAir, um sistema que pode ser integrado a aplicações móveis para permitir *offloading* de funcionalidades para servidores de alta capacidade e escalabilidade computacional, oferecendo devido suporte tanto para funcionalidades básicas como para aquelas que demandam um grande poder de processamento de forma eficiente e transparente para a aplicação.

[1] Em inglês: Android Interface Definition Language (AIDL).

Os focos de sua otimização podem ser divididos em lados cliente e servidor:

- O lado do cliente possui como foco de otimização tempo de execução e economia de energia.

  Para atingir este objetivo o sistema emprega uso de históricos de execução e consumo energético, em conjunto com parâmetros do ambiente no qual o dispositivo se encontra, para efetuar as decisões de *offloading* para cada funcionalidade.

- O lado do servidor possui como foco de otimização tempo e custo de execução.

  Para atingir este objetivo o sistema emprega até seis tipos de máquinas virtuais para *Android*, cada um com poder de processamento e capacidade de memória distintos, empregados para paralelizar as execuções de funcionalidades de acordo com o tipo de algoritmo utilizado, e se os parâmetros de entrada requerem uma quantidade significantemente grande de memória para o processamento.

MACS: Kovachev, Yu e Klamma (2012) desenvolveram o *framework* MACS, um sistema autoadaptativo que emprega particionamento de aplicações com baixo impacto computacional e *offloading* de execução das aplicações de forma fluída.

O foco de sua otimização é no custo energético, custo de transferência, uso de CPU e tempo de execução. Para atingir este objetivo, cada uma das abordagens é representada como uma função matemática, e a decisão final é obtida através da minimização da combinação dos valores obtidos pelas funções.

MAsCOT: Naqvi et al. (2016) desenvolveram o *framework* MAsCOT, um sistema para suporte à tomada de decisão auto-adaptativa de *offloading* que adequa-se eficientemente a cada tipo de aplicação e ao contexto de execução no qual se encontra.

O foco de sua otimização é no tempo de execução, trocas de dado entre cliente e servidor, e otimização de uso dos recursos disponíveis para a aplicação móvel. Para atingir este objetivo o *framework* modela todos os fatores pertinentes como uma rede de decisão dinâmica, que é um modelo de grafo acíclico direcionado, otimizando sua utilização através da teoria dos grafos e operações matemáticas.

Odessa: Ra et al. (2011) desenvolveram o *framework* Odessa, um sistema que oferece suporte à paralelização de processamento de dados em nível de *pipeline*, realizando *offloading* em nível de componentes. Utiliza como base o sistema de processamento distribuído de fluxo de dados Sprout (PILLAI et al., 2009).

O foco de sua otimização é em maximizar fluxo de processamento de dados e minimizar tempo gasto em cada janela de processamento, avaliando tempo de execução, paralelização de *pipeline* de processamento e latência entre dispositivo e servidor remoto em cada etapa de execução da aplicação, onde cada etapa é representada como nó de um grafo.

2.2.5 Análise e Comparação dos *Frameworks*

Prosseguindo com a discussão sobre *frameworks* de MCC, esta seção apresenta uma breve análise e comparação entre os nove *frameworks* anteriormente apresentados, organizados por objetivos estabelecidos para a otimização de tomada de decisão de *offloading*, parâmetros utilizados para efetuar a tomada de decisão, e se o *framework* armazena histórico de execução das funcionalidades para uso posterior nas tomadas de decisão. As comparações encontram-se resumidas na Tabela 2.2.

Tabela 2.2: Comparação Entre *Frameworks*

Framework	**Objetivos de Otimização**	**Representação dos Parâmetros de Otimização**	**Uso de Histórico**
MAUI	Economia de energia	Avaliação de histórico de execução	Sim
CloneCloud	Custo geral de execução	Custos expressos em funções matemáticas e otimizados por Programação Linear Inteira	Não
MpOS	Latência de rede	Avaliação de latência de rede	Não
AIOLOS	Tempo de execução, Latência de rede	Custos expressos por funções matemáticas	Sim
Cuckoo	Tempo de execução, Economia de energia	Avaliação por heurísticas e histórico de execução	Sim
ThinkAir	Tempo de execução, Custo de Execução, Economia de energia	Lado cliente: Avaliação por heurísticas, parâmetros do ambiente e históricos de execução e consumo energético Lado servidor: Tamanho de memória alocada durante execução	Sim
MACS	Custo de execução, economia de energia, custo de memória	Custos expressos em funções matemáticas e otimizados por Programação Linear Inteira	Não
MAsCOT	Tempo de execução, custo de execução, uso de recursos	Custos expressos em grafos e otimizados por Modelo Gráfico Probabilístico	Sim
Odessa	Taxa por tempo de execução, latência de rede e paralelismo de *pipeline*	Fluxo de execução expresso em grafo, avaliando cada nó por meio de funções matemáticas	Sim

Elaborada pelo autor.

Objetivos de Otimização

Dentre os trabalhos analisados, observou-se que os *frameworks* empregam principalmente os seguintes parâmetros a serem otimizados na execução de aplicações móveis:

- Tempo de execução da aplicação, adotado por 6 dos 9 *frameworks* analisados;
- Consumo energético pela execução da aplicação, adotado por 4 dos 9 *frameworks* analisados;

- Latência da conexão de dados entre dispositivo móvel e servidor remoto, adotado por 3 dos 9 *frameworks* analisados.

Embora existam outros objetivos de otimização também empregados para decisão de *offloading*, estes são aplicados por conjuntos menores de *frameworks* dentre estes estudados.

A relação entre os *frameworks* e seus respectivos parâmetros de otimização encontra-se ilustrada na Tabela 2.3.

Tabela 2.3: Relação dos parâmetros de otimização adotados por cada *framework*

Framework	Tempo de Execução	Consumo Energético	Latência	Outros Objetivos
MAUI		✓		
CloneCloud				Custo geral de execução
MpOS			✓	
AIOLOS	✓		✓	
Cuckoo	✓	✓		
ThinkAir	✓	✓		
MACS	✓	✓		Custo de memória utilizada
MAsCOT	✓			Custo de uso dos recursos
Odessa	✓		✓	Paralelização de *pipeline*

Elaborada pelo autor.

Para o uso de tempo de execução como parâmetro de otimização, AIOLOS, Cuckoo, ThinkAir, MACS, MAsCOT e Odessa realizam este uso.

Dentre os *frameworks* que adotam consumo energético como parâmetro de otimização, MAUI, Cuckoo, ThinkAir e MACS realizam esta abordagem.

Para a questão de minimização de latência de rede, MpOS, AIOLOS e Odessa abordam diretamente este parâmetro.

Há outros parâmetros de otimização também observados, como custo geral de execução (CloneCloud), custo de uso dos recursos disponíveis para a aplicação móvel (MAsCOT), custo de memória utilizada (MACS) e paralelização do *pipeline* de execução da aplicação (Odessa).

Representação dos Parâmetros de Otimização para Tomada de Decisão

Quanto mais parâmetros de otimização empregados na tomada de decisão de *offloading*, maior torna-se a complexidade na tomada de decisão, o que leva a busca por modelos mais robustos e sofisticados para representar estes parâmetros.

Dentre os trabalhos anteriormente analisados, pode-se notar que existem diversas abordagens para a representação interna de cada parâmetro de otimização.

- Por heurísticas, adotado por 3 dos 9 *frameworks* analisados;

- Por funções matemáticas, adotado por 4 dos 9 *frameworks* analisados;
- Por grafo, adotado por 3 dos 9 *frameworks* analisados;
- Por funções de custo, adotado por somente 1 dentre os 9 *frameworks* analisados.

A relação entre os *frameworks* e suas respectivas representações de parâmetros adotadas encontra-se ilustrada na Tabela 2.4.

Tabela 2.4: Relação das abordagens de representação dos parâmetros de otimização adotadas por cada *framework*

Nome	Heurísticas	Funções Matemáticas	Grafos	Funções de Custo
MAUI			✓	
CloneCloud		✓		
MpOS	✓			
AIOLOS		✓		✓
Cuckoo	✓			
ThinkAir	✓			
MACS		✓		
MAsCOT			✓	
Odessa		✓	✓	

Elaborada pelo autor.

Os *frameworks* MpOS, Cuckoo e ThinkAir adotam heurísticas predeterminadas para representar seus parâmetros de otimização e obter a decisão de *offloading*. MpOS somente avalia a latência da rede, realizando tomada de decisão baseando-se somente neste parâmetro. Cuckoo utiliza simples heurísticas próprias, avaliando histórico de tempo de execução, juntamente com informações sobre conexão com servidor, comparando diretamente cada invocação local e remota das funcionalidades para obter a decisão de *offloading*.

Para representar todos os seus parâmetros como funções matemáticas, CloneCloud, AIOLOS, MACS e Odessa adotam esta abordagem. Para obter a decisão final a partir de várias funções, CloneCloud e MACS empregam uso de Programação linear inteira para obter uma decisão que otimize as decisões de todas as funções, enquanto AIOLOS foca em otimizar um objetivo por vez. Já Odessa busca otimizar todos os objetivos simultaneamente, considerando situações onde ao menos um dos objetivos não é otimizado como estando sucetível a erros.

Para representar todos os seus parâmetros como grafo, MAUI, MAsCOT e Odessa adotam esta abordagem. Para obter a decisão final a partir dos grafos obtidos, MAUI emprega programação linear inteira para minimizar uso energético por toda a execução da aplicação móvel. MAsCOT emprega uma abordagem estocástica para a tomada de decisão através do uso de Modelos Gráficos Probabilísticos. Já Odessa.

Para formular as funções de custo, AIOLOS emprega regressão linear a partir das informações obtidas do histórico de execução.

Já o *framework* ThinkAir adota abordagens distintas para o lado cliente e para o lado servidor, baseando-se somente em heurísticas próprias. A abordagem empregada pelo lado do cliente assemelha-se à abordagem empregada por Cuckoo, avaliando históricos de execução e informações do ambiente, comparando diretamente cada invocação local e remota das funcionalidades para obter a decisão de *offloading*. Do lado do servidor, o *framework* adapta sua execução conforme o tamanho de memória requisitado reativamente, sem qualquer forma de predição proativa.

Uso de Histórico de Execução para Tomada de Decisão

O uso de histórico de execução das aplicações nas tomadas de decisão para *offloading* de funcionalidades oferece predições de execução mais fiéis do que predições que não envolvem histórico, permitindo que o *framework* possa inferir os parâmetros de otimização locais e remotos de execução das aplicações de forma mais precisa e, com base nas predições obtidas, otimizando a execução das aplicações eficientemente (NARAYANAN; FLINN; SATYANARAYANAN, 2000).

Dentre os trabalhos anteriormente citados, MAUI, AIOLOS, Cuckoo, ThinkAir, MAsCOT e Odessa armazenam histórico de execução de cada funcionalidade e utilizam para tomadas de decisão de *offloading*. Enquanto AIOLOS, Cuckoo, ThinkAir e Odessa armazenam histórico de execução na forma de *logs*, MAUI e MAsCOT o armazena em forma de grafo.

2.3 Considerações Finais do Capítulo

Este capítulo apresentou revisão das principais áreas de estudo utilizadas para a confecção deste trabalho.

A seção de sistemas auto-adaptativo apresentou uma introdução básica e suas principais características. Também apresentou-se um modelo de desenvolvimento emergente dentro de sistemas auto-adaptativos, chamado de sistemas autoconscientes e auto-expressivos.

A seção referente à área de computação móvel nas nuvens apresentou uma introdução básica referente, relação de ciência de contexto com a área em questão, além de introdução e análise de alguns dos principais *frameworks* existentes na literatura segundo a ótica de tomada de decisão para *offloading* de componentes de aplicações móveis e otimização de performance de execução.

Com base neste capítulo, pode-se notar que há uma grande variedade de formas de representação para os parâmetros de otimização assim como para os meios de tomada de decisão de *offloading*. E estas representações são diretamente relacionados ao meio como os *frameworks* são acoplados às aplicações móveis aos quais será realizado o suporte.

Embora exista um vasto número de *frameworks* desenvolvidos na área de MCC, observam-se poucos *frameworks* que relatam uso explícito de arquitetura para sistemas autoadaptativos, com MAsCOT sendo um dos raros exemplos dentro da literatura a adotar seu uso por meio do modelo de referência auto-adaptativa MAPE-K - Monitorar-Analisar-Planejar-Executar sobre uma partição compartilhada de Conhecimento. Para este fim CoSMOS busca contribuir ao utilizar arquitetura de sistemas autoconscientes e autoexpressivos como base para seu desenvolvimento.

Nos próximos capítulos será apresentado o modelo CoSMOS e as etapas de planejamento, desenvolvimento e avaliação.

[1] Em inglês: Monitor-Analyze-Plan-Execute over a shared Knowledge.

Capítulo 3

PROJETO DO COSMOS

Com base nos estudos de literatura realizados anteriormente, deu-se início à fase de projeto e elaboração do presente trabalho de criação do modelo CoSMOS (*Context-Sensitive Model for Offloading System*), um modelo desenvolvido para a tomada de decisão de migração de dados de uma dada aplicação móvel.

Para desenvolver o projeto do modelo CoSMOS foram adotadas as seguintes etapas:

- Escolha da forma de suporte a ser realizada para aplicações móveis;
- Levantamento de requisitos funcionais e não-funcionais;
- Seleção da arquitetura de sistemas autoconscientes e auto-expressivos com base nos requisitos levantados na etapa anterior;
- Com base da forma de suporte, requisitos de funcionamento e arquitetura selecionada, desenvolver fluxo para tomada de decisão.

Nas próximas seções deste capítulo, cada uma das etapas elicitadas acima serão melhor explicadas e detalhadas.

3.1 Suporte à Execução de Aplicações Móveis

Inicialmente foi estudada a forma que seria realizado o suporte à execução de aplicações para efetuar os estudos necessários para este trabalho.

Para a elaboração deste trabalho foram levantadas duas opções: adaptar um *framework* de suporte já existente na literatura, ou desenvolver um *framework* novo.

Com o objetivo de focar o desenvolvimento deste trabalho somente na modelagem de tomada de decisão, optou-se por utilizar um *framework* já existente na literatura como base de desenvolvimento ao invés de conceber um novo *framework*. Ao aproveitar componentes de infraestrutura de comunicação e acoplamento às aplicações já existentes do *framework* base, permite-se realizar o trabalho com foco somente no desenvolvimento de modelo de tomada de decisão, que é o objetivo deste trabalho.

Com a definição da abordagem de desenvolvimento, iniciou-se a etapa de levantamento de *frameworks* existentes que pudessem ser utilizados para a realização deste trabalho. Dentre os trabalhos analisados, somente dois destes estavam disponíveis para acesso a uso e avaliação: Cuckoo (KEMP et al., 2012) e MpOS (COSTA et al., 2015).

Cuckoo oferece suporte para aplicações desenvolvidas somente através da IDE Eclipse versão 3.8 Juno, não oferecendo suporte para versões posteriores ou outras IDEs, limitando severamente seu uso para aplicações móveis mais recentes.

Já MpOS oferece suporte para aplicações desenvolvidas tanto por Eclipse como pelo Android Studio, independentemente da versão de IDE utilizada. Este fator, aliado ao meio de acoplamento às aplicações móveis por meio de marcação no código-fonte, oferece benefícios a uma gama maior de aplicações.

Com base nestes pontos optou-se por utilizar o *framework* MpOS como base na confecção deste trabalho.

3.2 Arquitetura do Modelo CoSMOS

Após as escolhas da forma de desenvolvimento do projeto e da abordagem para representação dos parâmetros de otimização, iniciou-se a fase de aplicação dos padrões de projeto para sistemas autoconscientes e auto-expressivos e a escolha do modelo de arquitetura de sistema mais adequado para o desenvolvimento deste trabalho.

3.2.1 Sensores e Atuadores

A partir de estudo e análise do *framework* MpOS foram elicitadas as fontes de informação para o *framework*, categorizados como sensores, e meios de externalização de informações a partir do *framework*, categorizados como atuadores.

Sensores e atuadores foram classificados também em dois tipos: internos e externos. Enquanto sensores e atuadores internos tratam de informações provenientes das chamadas de métodos e de dados relevantes que podem ser acessados a partir da SDK do Android, sensores e atuadores externos são relativos ao ambiente de execução externo ao próprio dispositivo que afetem diretamente a aplicação. No caso foi considerado somente o servidor remoto de suporte ao *offloading*.

Sensores internos:

- Chamada de método da aplicação;
- Tipo de conexão estabelecida;
- Medição de amperagem média da corrente da bateria do dispositivo;
- Medição de voltagem da bateria do dispositivo;
- Pedido de medição da conexão estabelecida com o servidor remoto.

Sensores externos:

- Conexão com servidor;
- Largura de banda de conexão entre dispositivo e servidor remoto;
- Dado de processamento remoto de método chamado.

Atuador interno:

- Execução da aplicação.

Atuadores externos:

- Busca por servidor remoto;
- Dados para *offloading* de funcionalidade para servidor remoto;
- Pedido para medição de largura de banda entre dispositivo móvel e servidor remoto.

3.2.2 Levantamento de Requisitos e Restrições

Tomando como base os sensores e atuadores levantados em em 3.2.1, foram elicitados requisitos funcionais e restrições considerados importantes para a elaboração de um modelo de tomada de decisão multi-objetiva auto-adaptativa. Não foram levantados requisitos não-funcionais para este projeto.

Os requisitos e as restrições levantados se encontram dispostos respectivamente pelas Tabelas 3.1 e 3.2.

Tabela 3.1: Requisitos funcionais do modelo CoSMOS

Identificação	**Descrição**
RF1	O modelo deve ser capaz de inferir *a priori* se será vantajoso realizar *offloading* de determinada chamada a método.
RF2	O modelo deve ser capaz de inferir o tempo de execução e o consumo energético para cada chamada a método.
RF3	O modelo deve salvar informações relativas a cada chamada de método.
RF4	O modelo deve oferecer mais de uma abordagem para a tomada de decisão, para prover previsões de decisão mais precisas em casos de possível conflito.
RF5	O modelo deve ser capaz de refinar suas abordagens de tomada de decisão iterativamente, ou seja, melhorar a decisão conforme mais execuções forem realizadas.
RF6	O modelo deve ser capaz de analisar o grau de variação entre previsões de execução de chamadas e valores reais.
RF7	O modelo deve ser capaz de alterar suas prioridades de tomada de decisão conforme nível energético do dispositivo.
RF8	O modelo deve ser capaz de inferir diferentemente para servidores e meios de comunicação distintos
RF9	O modelo deve ser capaz de carregar dados necessários para tomada de decisão de acordo com o servidor ao qual se encontrar conectado.

Elaborado pelo autor.

Tabela 3.2: Restrições do modelo CoSMOS

Identificação	Descrição
RT1	O modelo deverá reter a granularidade e o suporte do *framework* base adotado.
RT2	O modelo só terá acesso a bibliotecas Android para realizar suas operações, sem necessidade de instalar outras bibliotecas;

Elaborado pelo autor.

3.2.3 Seleção de Padrão de Arquitetura

Após elicitar os requisitos e restrições considerados importantes pela seção 3.2.2 foi realizado estudo sobre as seis aptidões de sistemas autoconscientes e auto-expressivos listadas na Tabela 2.1, e suas respectivas relevâncias dentro do modelo CosMOS. Para elicitar a relevância de cada aptidão foi utilizado um questionário formado por cinco perguntas, elaborado por Chen et al. (2014). Este questionário foi aplicado para cada uma das seis aptidões, com enunciados dispostos abaixo:

1. Qual a importância da aptidão dentro do contexto do problema?
2. Quais os requisitos funcionais que influenciam esta aptidão?
3. Quais os requisitos não-funcionais que influenciam esta aptidão?
4. Quais as restrições que influenciam esta aptidão?
5. Esta aptidão é benéfica ou essencial?

Enquanto a pergunta 1 possui resposta descritiva sobre a aplicação de cada aptidão dentro do modelo, as perguntas 2, 3 e 4 remetem aos requisitos e restrições listados na seção 3.2.2, dispondo somente os relevantes para cada aptidão. A pergunta 5 verifica se a aptidão é essencial para o desenvolvimento, ou se somente agrega uma funcionalidade extra para o modelo. Como nenhuma restrição não-funcional foi levantada para este trabalho, as respostas à pergunta 3 encontram-se vazias.

O questionário visa oferecer uma abordagem sistemática para a seleção e aplicação dos padrões de projeto para sistemas autoconscientes e auto-expressivos. O questionário aplicado às aptidões de consciência de estímulo, interação, tempo, objetivo, auto-expressão e meta-autoconsciência encontra-se disposto, respectivamente, nas Tabelas de 3.3 a 3.8:

Tabela 3.3: Questionário para decisão de inclusão da aptidão de consciência de estímulo

Consciência de Estímulo	
Qual a importância da aptidão dentro do contexto do problema?	Realizar tomada de decisão de *offloading* de funcionalidade baseando-se somente nas informações referentes à chamada de método e dimensões dos dados de entrada.
Quais os requisitos funcionais que influenciam esta aptidão?	RF1, RF2, RF4, RF5, RF6, RF8
Quais os requisitos não-funcionais que influenciam esta aptidão?	-
Quais as restrições que influenciam esta aptidão?	RT1, RT2
Esta aptidão é benéfica ou essencial?	Essencial

Baseado em Chen et al. (2014).

Tabela 3.4: Questionário para decisão de inclusão da aptidão de consciência de interação

Consciência de Interação	
Qual a importância da aptidão dentro do contexto do problema?	Realizar tomada de decisão de *offloading* de funcionalidade baseando-se nas informações referentes à chamada de método, dimensões dos dados de entrada e estado da conexão entre dispositivo cliente e servidor remoto.
Quais os requisitos funcionais que influenciam esta aptidão?	RF1, RF2, RF4, RF5, RF8
Quais os requisitos não-funcionais que influenciam esta aptidão?	-
Quais as restrições que influenciam esta aptidão?	RT1, RT2
Esta aptidão é benéfica ou essencial?	Essencial

Baseado em Chen et al. (2014).

Tabela 3.5: Questionário para decisão de inclusão da aptidão de consciência de tempo

Consciência de Tempo	
Qual a importância da aptidão dentro do contexto do problema?	Armazenar e recuperar informações relativas às execuções locais e remotas de funcionalidades da aplicação.
Quais os requisitos funcionais que influenciam esta aptidão?	RF1, RF3, RF5, RF8, RF9
Quais os requisitos não-funcionais que influenciam esta aptidão?	-
Quais as restrições que influenciam esta aptidão?	RT1, RT2
Esta aptidão é benéfica ou essencial?	Essencial

Baseado em Chen et al. (2014).

Tabela 3.6: Questionário para decisão de inclusão da aptidão de consciência de objetivo

Consciência de Objetivo	
Qual a importância da aptidão dentro do contexto do problema?	Controlar prioridade de adotar ou não *offloading* durante tomada de decisão.
Quais os requisitos funcionais que influenciam esta aptidão?	RF1, RF2, RF5, RF6, RF7
Quais os requisitos não-funcionais que influenciam esta aptidão?	-
Quais as restrições que influenciam esta aptidão?	RT1, RT2
Esta aptidão é benéfica ou essencial?	Essencial

Baseado em Chen et al. (2014).

Tabela 3.7: Questionário para decisão de inclusão de auto-expressão

Auto-Expressão	
Qual a importância da aptidão dentro do contexto do problema?	Realizar interface entre modelo de tomada de decisão e *framework* de execução de aplicações móveis.
Quais os requisitos funcionais que influenciam esta aptidão?	RF1, RF3, RF4, RF5, RF6, RF7, RF8, RF9
Quais os requisitos não-funcionais que influenciam esta aptidão?	-
Quais as restrições que influenciam esta aptidão?	RT1, RT2
Esta aptidão é benéfica ou essencial?	Essencial

Baseado em Chen et al. (2014).

Tabela 3.8: Questionário para decisão de inclusão da aptidão de meta-autoconsciência

Meta-Autoconsciência	
Qual a importância da aptidão dentro do contexto do problema?	Gerenciar uso das demais camadas, delegar chamada devidamente para as camadas responsáveis.
Quais os requisitos funcionais que influenciam esta aptidão?	RF1, RF3, RF4, RF5, RF7, RF8, RF9
Quais os requisitos não-funcionais que influenciam esta aptidão?	-
Quais as restrições que influenciam esta aptidão?	RT1
Esta aptidão é benéfica ou essencial?	Essencial

Baseado em Chen et al. (2014).

A partir dos seis questionários realizados nota-se que todas as aptidões foram marcadas como essenciais para a elaboração do modelo de tomada de decisão autoconsciente. Sendo assim optou-se por utilizar como base de desenvolvimento o padrão de arquitetura autoconsciente e auto-expressivo, que encontra-se exemplificado na Figura 2.2.

3.3 Definição do Modelo CoSMOS

A partir da análise sistemática dos requisitovs relevantes e das aptidões marcadas como essenciais, deu-se a divisão dos papéis necessários para realização da tomada de decisão de *offloading* e a elaboração do fluxo de execução a ser adotado para o modelo, tomando como base o fluxo básico estabelecido pela literatura na Seção 2.2.2 e exemplificado na Figura 2.4.

Para a elaboração do modelo de tomada de decisão foram considerados os seguintes papéis necessários:

- Tomada de decisão para cada parâmetro;
- Solução de conflitos entre parâmetros e unificação de decisão;
- Diretivas principais para definir prioridades na tomada de decisão;
- Acesso e persistência de dados;
- Comunicação entre CoSMOS e MpOS.

A relação entre os papéis de tomada de decisão levantados e as aptidões de sistemas autoconscientes encontra-se ilustrado pela Tabela 3.9.

Tabela 3.9: Relação entre aptidões levantadas e as tarefas para tomada de decisão

Papel para Tomada de Decisão	**Aptidão em Sistemas Autoconscientes**
Tomada de decisão	Consciência de Estímulo Consciência de Interação
Solução de conflitos entre parâmetros e unificação de decisão	Meta-Autoconsciência
Diretivas principais para definir prioridades na tomada de decisão	Consciência de Objetivo Meta-Autoconsciência
Acesso e persistência de dados	Consciência de Tempo
Comunicação entre CoSMOS e MpOS	Auto-Expressão

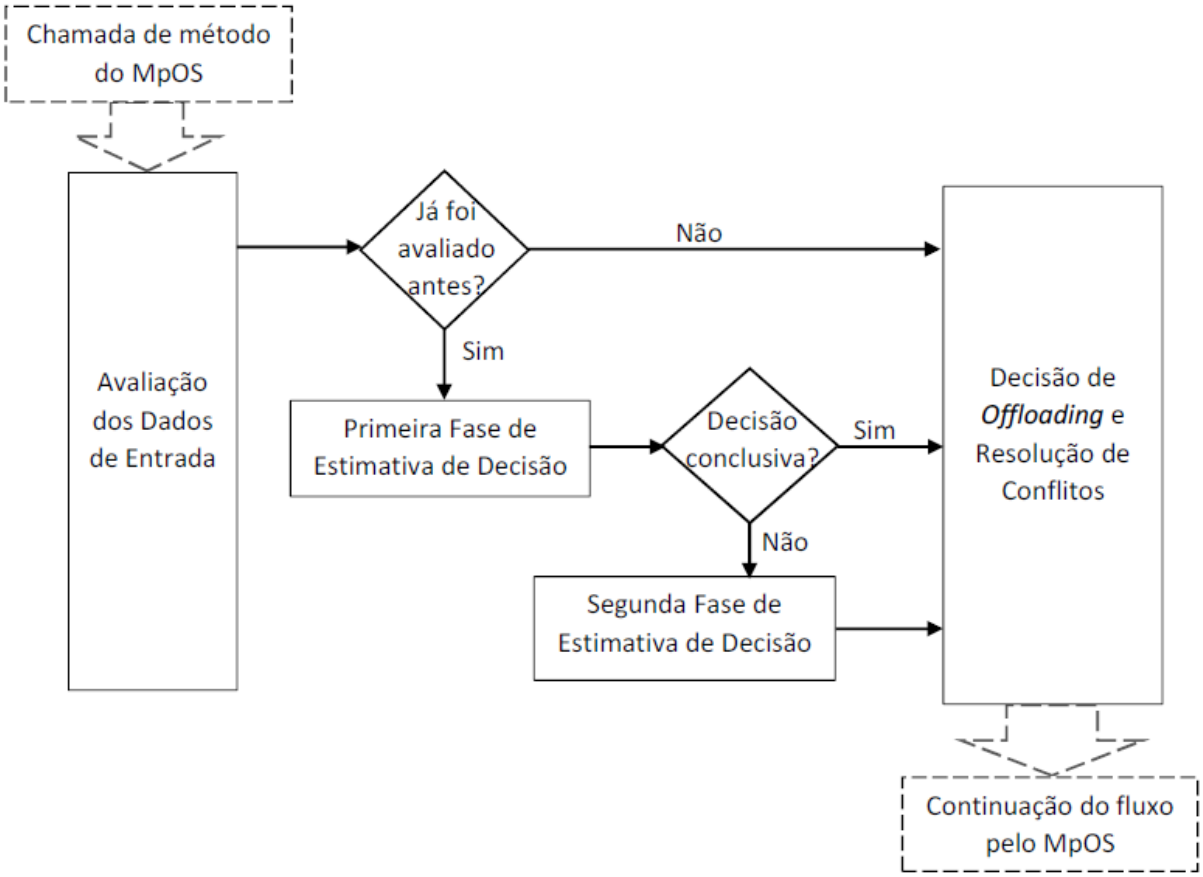
Elaborada pelo autor.

Optou-se por atribuir duas aptidões para a tomada de decisão com o propósito de diminuir o *overhead* computacional, que possui como principais fatores o monitoramento do estado do ambiente de execução e processamento de informações (AHMED et al., 2015; FERNANDO; LOKE; RAHAYU, 2013).

Com o propósito de minimizar o impacto do monitoramento buscou-se utilizar a aptidão de consciência de estímulo, com a consciência de interação oferecendo suporte para melhorar a acuidade das decisões em casos com pouca diferença entre as performances de execução local e remota das aplicações.

A partir dos papéis para tomada de decisão de *offloading*, aptidões de sistemas autoconscientes relacionados, e tomando como base o fluxo básico de *offloading*, propôs-se um modelo para tomada de decisão para CoSMOS. O fluxo de execução deste modelo encontra-se ilustrado na Figura 3.1.

Figura 3.1: Fluxograma do modelo CoSMOS



Elaborado pelo autor.

3.4 Considerações Finais do Capítulo

Este capítulo apresentou o planejamento do modelo CoSMOS, utilizando como etapas o levantamento de requisitos, levantamento de aptidões de sistemas autoconscientes e auto-expressivos, elicitação de papéis necessários para realização de tomada de decisão para *offloading* computacional, e elaboração de fluxo de execução com base no fluxo básico de sistemas de suporte a computação móvel nas nuvens.

As linhas-guia dispostas através dos padrões de arquitetura elaborados por Chen et al. (2014) auxiliaram durante a elaboração deste trabalho.

No próximo capítulo serão apresentados visão geral da implementação do modelo e fluxos principais de suas operações.

Capítulo 4

DESENVOLVIMENTO DO COSMOS

Após definidas a abordagem de desenvolvimento do modelo, a escolha dos parâmetros de otimização, suas representações dentro do sistema e a arquitetura adotada para sua elaboração, segue-se para a etapa de implementação do modelo.

Este capítulo propõe-se a apresentar a implementação do modelo CoSMOS e funcionamento dos componentes constituintes.

4.1 Implementação do Modelo CoSMOS

A implementação do modelo CoSMOS foi realizada utilizando como base para desenvolvimento a arquitetura do padrão de meta-autoconsciência e auto-expressão para sistemas autoconscientes e auto-expressivos, discutido na sessão 3.2.3.

A base principal de operações do modelo foi elaborada sobre a classe DecisionController, responsável por centralizar todas as operações relacionadas à tomada de decisão de *offloading* e realizar interface entre o *framework* MpOS e CoSMOS.

Dentro do modelo, com base da arquitetura levantada, foram elicitados quatro conjuntos de operações necessárias para a tomada de decisão de *offloading*:

- Análise de comportamento de execução da aplicação e estimação na primeira fase de validação;
- Centralizar informações sobre dispositivo e conexão entre dispositivo e servidor remoto para estimação da segunda fase de validação;
- Acesso e persistência de dados, a serem utilizados em futuras tomadas de decisão;

- Resolução de conflito de decisão entre os objetivos de otimização.

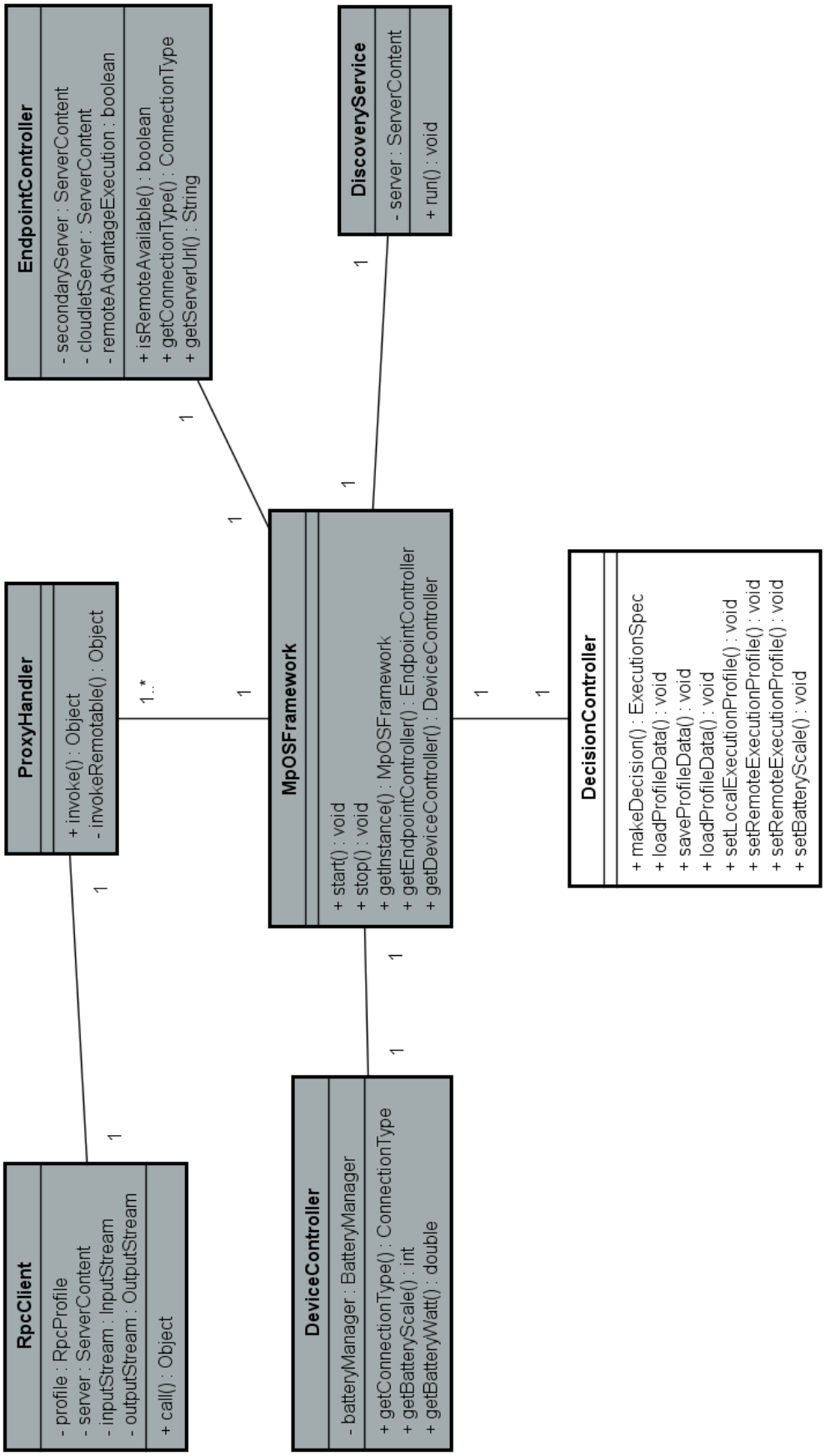
Para cada um dos conjuntos de operações foram desenvolvidas classes. Para realizar interface entre o *framework* base e o modelo, há a classe DecisionController, desenvolvida seguindo o padrão de projetos Fachada (GAMMA et al., 1995).

Esta classe possui quatro operações essenciais:

- Logo ao iniciar a aplicação móvel e o MpOS, para carregar os dados essenciais para as operações do CoSMOS, é invocado o método loadProfileData(), responsável por invocar todos os métodos que carregam as informações necessárias de acordo com a aplicação e o servidor remoto conectado;
- Durante a interceptação da chamada dos métodos marcados, para efetuar tomada de decisão de *offloading*, é invocado o método makeDecision(), responsável por invocar os métodos de tomada de decisão;
- Logo após a execução do método invocado da aplicação, logo antes de voltar ao fluxo original, são coletados as informações de execução, e passados para CoSMOS através dos métodos setLocalExecutionProfile() e setRemoteExecutionProfile();
- Ao encerrar a aplicação, durante o encerramento do MpOS é invocado o método saveProfileData(), responsável por invocar todos os métodos de persistência de dados.

A Figura 4.1 apresenta o relacionamento da classe DecisionController com o *framework* MpOS através de um diagrama de classes UML.

Figura 4.1: Diagrama de classes do modelo CoSMOS



Elaborado pelo autor.

É importante mencionar que o Controller utilizado no nome desta e das demais classes do CoSMOS implica somente que cada classe é controladora das operações a ela designadas, não possuindo relação alguma com o padrão de projetos MVC - Modelo-Visão-Controlador[1].

4.1.1 Implementação da Arquitetura Autoconsciente e Autoexpressiva

A relação entre as camadas da arquitetura para sistema autoconsciente e auto-expressivo, as classes implementadas para o CoSMOS e seus respectivos papeis dentro do modelo encontra-se ilustrada pela Tabela 4.1.

Tabela 4.1: Relação entre as camadas da arquitetura de sistemas autoconscientes e auto-expressivos e classes desenvolvidas para o modelo CoSMOS

Camada em SAS	Classe no CoSMOS	Descrição
Estímulo	EstimationController	Operações relacionadas à primeira fase de validação, com estimação baseada somente no parâmetro de entrada;
Interação	InteractionController	Operações relacionadas à segunda fase de validação, com estimação baseada no parâmetro de entrada, velocidade de transferência de dados e RTT - Tempo de Ida e Volta entre dispositivo e servidor remoto[2];
Tempo	PersistenceController	Operações relacionadas a acesso e armazenamento de dados para estimação;
Objetivo	VeredictController	Responsável pela tomada de decisão final a partir das decisões dos dois parâmetros de otimização;
Meta-Autoconsciência	DecisionController	Gerencia troca de informações entre as camadas conforme necessário;
Auto-Expressão	DecisionController	Operações de comunicação entre CoSMOS e MpOS.

Elaborado pelo autor.

[1] Em inglês: Model-View-Controller.
[2] Em inglês: Round-Trip Time.

4.1.2 Parâmetros de Entrada

Para obter acesso às informações de bateria, tipo de conexão estabelecida com servidor remoto, endereço do servidor e taxa de transferência de dados entre dispositivo e servidor, as classes DeviceController e EndpointController originárias do *framework* MpOS foram modificadas, com o propósito de manter modularidade. Enquanto a classe DeviceController provê informações sobre estado da bateria e tipo de conexão com servidor, a classe EndpointController provê informações sobre taxa de transferência de dados e endereço do servidor remoto.

Para medir tempo de execução cronometra-se o tempo de execução de cada chamada de método na classe ProxyHandler do *framework* MpOS, seja executado localmente ou migrado para execução em servidor remoto.

Para obter as dimensões dos parâmetros de cada método da aplicação utiliza-se a classe Java ByteArrayStream para ler cada um deles e obter tais medidas. Optou-se por seu uso devida à ausência de ferramentas disponíveis pela API Android para obter as dimensões dos parâmetros de entrada até a presente versão.

4.1.3 Acesso e Persistência de Dados

O principal intermediador de requisições e persistência dos dados é a classe *Persistence Controller*, que recebe as requisições de DecisionController e as delega para a classe responsável por acessar os dados na base *SQLite* de acordo com o tipo de dado requisitado.

A classe PersistenceController possui ao todo 18 métodos, que podem ser classificados de acordo com o tipo de informações requisitadas:

- Para acesso às informações referentes ao servidor remoto são utilizados os métodos getServerProfile() e updateServerProfile();
- Para acesso às informações referentes a cada método da aplicação móvel e seus respectivos parâmetros é utilizado o método getMethodProfileByName();
- Para acesso às informações referentes à estimação de tempo de execução são utilizados os métodos getLocalFunctionMap(), getRemoteFunctionMap() e updateFunctionMap();
- Para acesso às informações referentes à estimação de consumo energético são utilizados os métodos getLocalBatteryMap(), getRemoteBatteryMap() e updateBatteryMap();

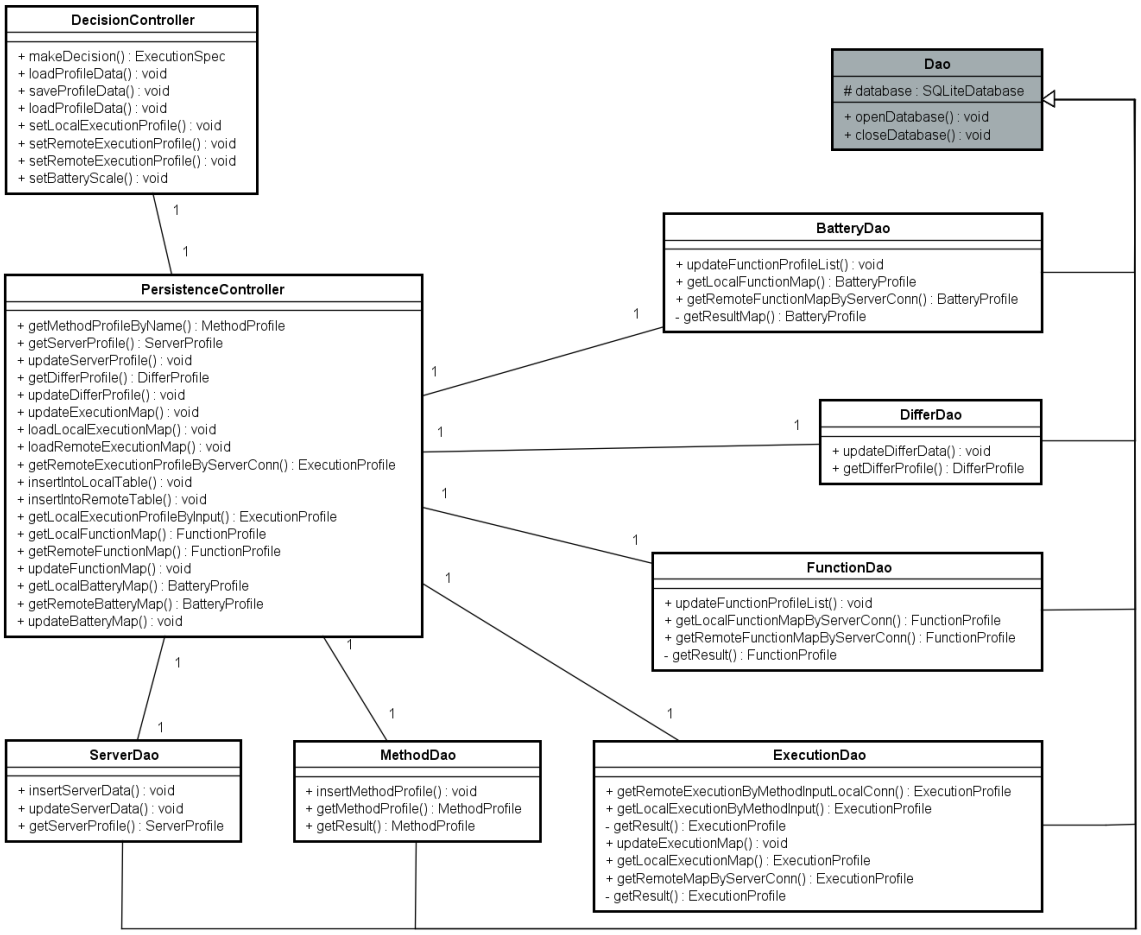
- Para acesso às informações referentes aos intervalos de estimação para uso da segunda fase de validação em tempo de execução e consumo energético são utilizados os métodos getDifferProfile() e updateDifferProfile();
- Para acesso às informações referentes ao histórico de tempo de execução e consumo energético são utilizados os métodos loadLocalExecutionProfile(), loadRemoteExecutionProfile(), getLocalExecutionProfileByInput(), getRemoteExecutionProfileByServerConn(), insertIntoLocalTable(), insertIntoRemoteTable() e updateExecutionMap().

Para realizar o acesso à base dados há seis classes distintas, responsáveis por seis tipos de informações distintas, que são utilizadas para a tomada de decisão de *offloading*. Estas classes são derivadas da classe de persistência Dao, que é originária do *framework* MpOS.

- ServerDao: responsável por acessar informações referentes ao servidor remoto;
- MethodDao: responsável por acessar informações referentes aos métodos da aplicação marcados como candidatos a *offloading*;
- FunctionDao: responsável por acessar informações referentes à estimação de tempo de execução das chamadas de método;
- BatteryDao: responsável por acessar informações referentes à estimação de consumo energético das chamadas de método;
- DifferDao: responsável por acessar informações referentes aos intervalos de estimação para uso da segunda fase de validação;
- ExecutionDao: responsável por acessar informações referentes ao histórico de execução dos métodos.

A Figura 4.2 apresenta o relacionamento das classes DecisionController, PersistenceController e as responsáveis pela persistência de cada tipo de dado utilizado pelo CoSMOS através do diagrama de classes UML.

Figura 4.2: Diagrama de classes para salvar dados de execução utilizados pelo modelo CoSMOS



Elaborado pelo autor.

4.1.4 Tomada de Decisão

Para realizar a tomada de decisão, o ciclo completo de execução do modelo CoSMOS pode ser dividido em três etapas:

- Etapa inicial, de carregamento de informações essenciais para realização das estimativas de parâmetros. Esta é executada ao iniciar a aplicação e o *framework*;
- Etapa principal, de estimativa de parâmetros de otimização e tomada de decisão de *offloading* das funcionalidades. Esta é realizada durante a execução da aplicação;
- Etapa final, de persistência das informações relacionadas às execuções realizadas. Esta é realizada durante o encerramento da aplicação

Durante as próximas seções serão apresentadas cada uma das três etapas elicitadas.

4.2 Etapa de Carga de Informações

A etapa de carga de informações é essencial para que o CoSMOS possa realizar a estimativa dos parâmetros de otimização e a tomada de decisão de *offloading*, pois é nesta etapa na qual todos os dados essenciais serão carregados.

Esta etapa pode ser também dividida em duas atividades: (i) carga de informações relativas à estimativa de execuções locais; e (ii) carga de informações relativas à estimativa remotas.

4.2.1 Informações de Execução Local

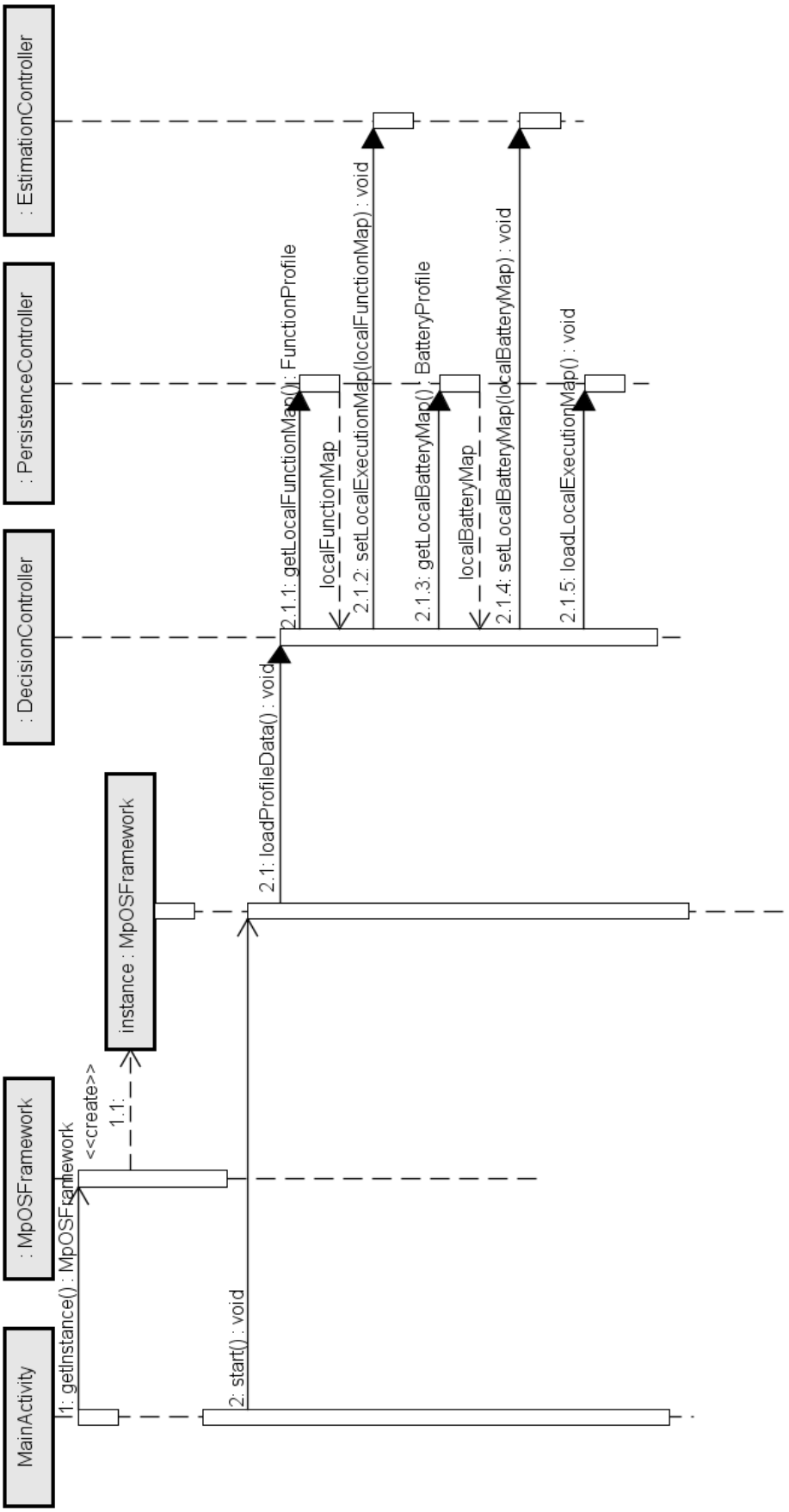
Esta subetapa é realizada logo ao iniciar a aplicação, juntamente à carga de todas as classes essenciais para a execução do MpOS, pois é independente do servidor remoto conectado ao *framework* e ao tipo de conexão estabelecida.

Ao iniciar a aplicação móvel, a classe MainActivity invoca o método start(), responsável por carregar os recursos necessários para o funcionamento do *framework*. Após carregar todos os componentes principais do *framework*, este invocará o método loadProfileData() da classe DecisionController, responsável por carregar todos os parâmetros para estimação de execução local e tomada de decisão com base nas execuções locais realizadas anteriormente.

Estes parâmetros são obtidos através das chamadas getLocalExecutionMap() e getLocalBatteryMap() da classe PersistenceController, e carregados para a classe EstimationController.

A Figura 4.3 apresenta a seqüência de operações adotada pelo modelo CoSMOS para realizar o carregamento de dados relativos às informações de execuções da aplicação móvel feitas localmente, representando através de um diagrama de sequência UML.

Figura 4.3: Diagrama de sequência de carregamento de dados de execução local para o *framework*



Elaborado pelo autor.

4.2.2 Informações de Execução Remota

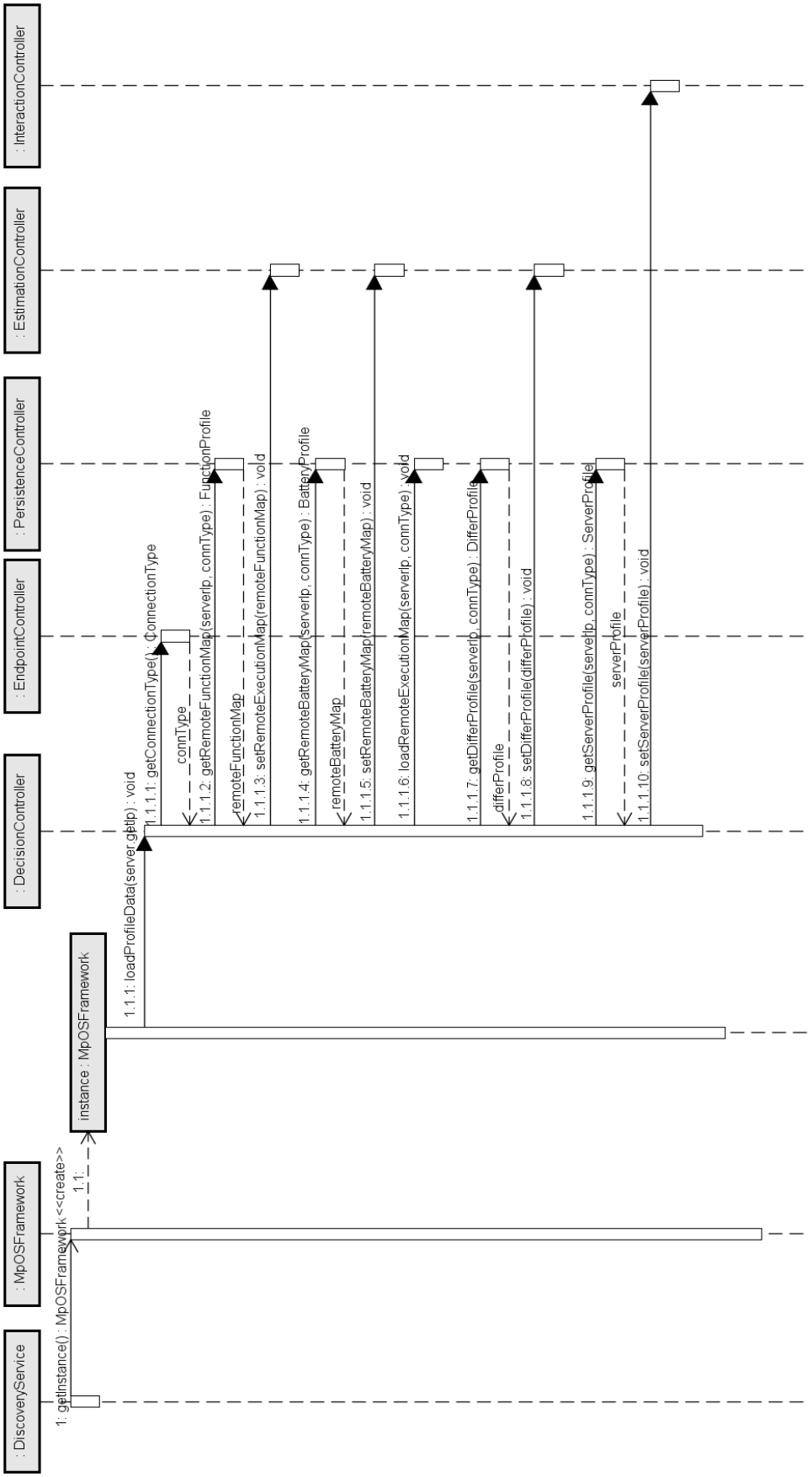
Diferentemente da subetapa relativa às informações de execução local, esta subetapa de execução remota é realizada somente após estabelecer conexão entre a aplicação e o servidor remoto. A partir das informações de endereço do servidor e do tipo de conexão estabelecida, esta subetapa carregará as informações necessárias para realizar a estimativa de parâmetros de otimização para execuções remotas.

Após o componente DiscoveryService do MpOS estabelecer conexão com uma plataforma de apoio remoto, este invocará o método loadProfileData() do componente DecisionController, método este que será responsável por carregar todos os parâmetros para a estimativa de execução remota e a tomada de decisão com base nas execuções remotas realizadas anteriormente para o servidor conectado.

Estes parâmetros são obtidos através das chamadas getRemoteExecutionMap() e getRemoteBatteryMap() do componente PersistenceController com base no IP do servidor de suporte e no tipo de conexão estabelecida entre o dispositivo móvel e o servidor nas nuvens. Os parâmetros, após obtidos, serão passados para a classe EstimationController.

A Figura 4.4 apresenta a seqüência de operações adotada pelo modelo CoSMOS para realizar a carga de dados relativos às informações de execuções da aplicação móvel feitas remotamente em um servidor nas nuvens, representando através de um diagrama de sequência UML.

Figura 4.4: Diagrama de sequência de carga de dados de execução remota para o *framework*



Elaborado pelo autor.

4.3 Etapa de *Offloading*

A etapa de *offloading* é a etapa principal do CoSMOS, responsável por decidir se determinada funcionalidade invocada pela chamada da aplicação móvel será executada localmente no dispositivo ou enviada para execução remota em servidor externo.

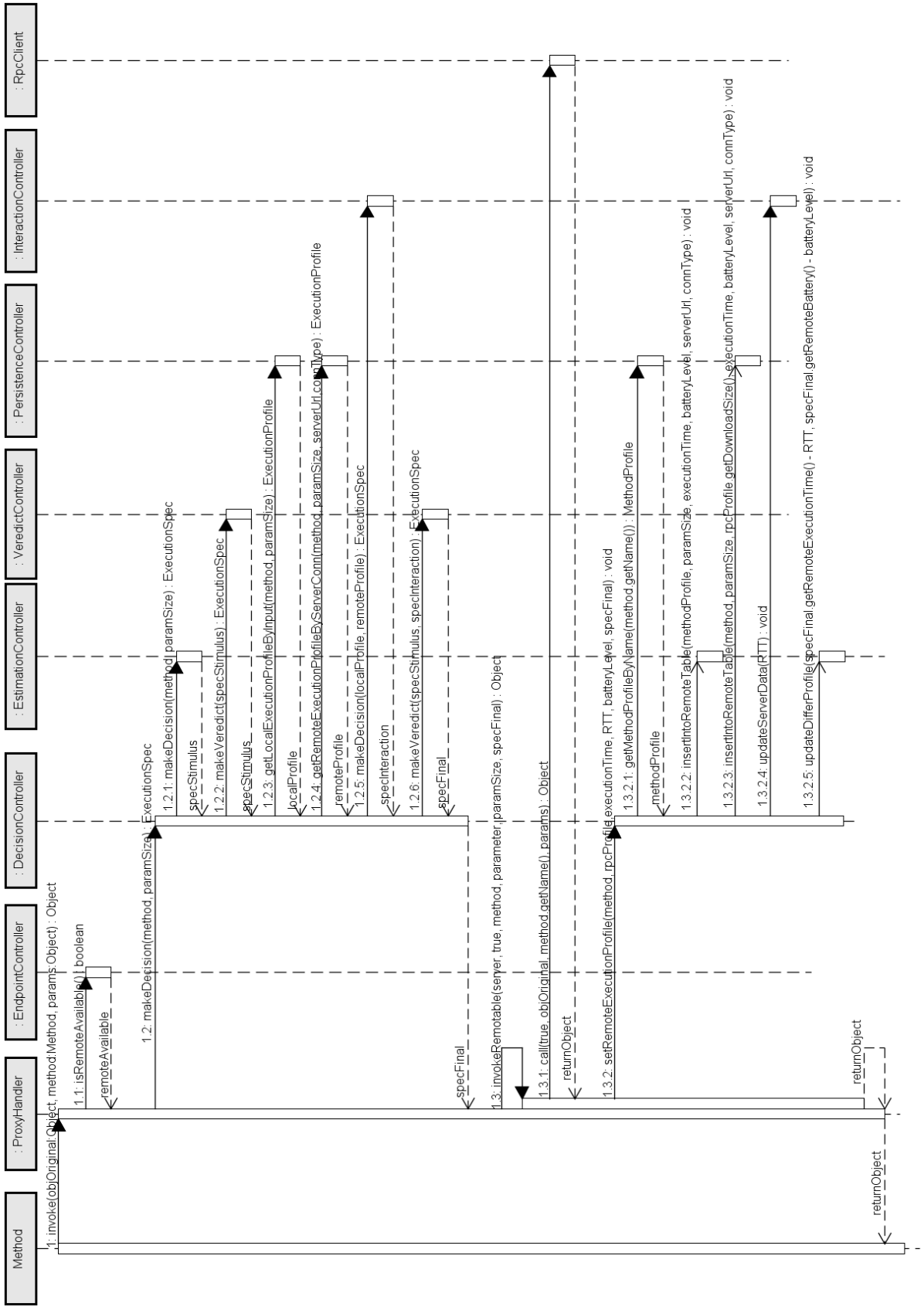
Dentro do modelo, para definir se dada funcionalidade já foi anteriormente executada para dado(s) parâmetros(s) de entrada anteriormente, o modelo grava o intervalo de dimensões de parâmetros para cada funcionalidade. Este intervalo é definido pelas dimensões máxima e mínima de parâmetros previamente executados pela funcionalidade.

Cada um dos parâmetros de otimização é responsável por monitorar e armazenar os valores de intervalo de parâmetros de entrada já analisados, que serão utilizados em suas respectivas tomadas de decisão de *offloading*.

A Figura 4.5 apresenta a seqüência de operações adotada por MpOS para tomada de decisão de *offloading* por meio do CoSMOS e realizar *offloading* de uma funcionalidade para o servidor remoto, representando através de um diagrama de sequência UML.

Nas próximas subseções serão apresentadas as duas fases de validação, e também a coleta de dados relativos às execuções realizadas.

Figura 4.5: Diagrama de sequência para *offloading* de funcionalidade



Elaborado pelo autor.

4.3.1 Etapa de Validação

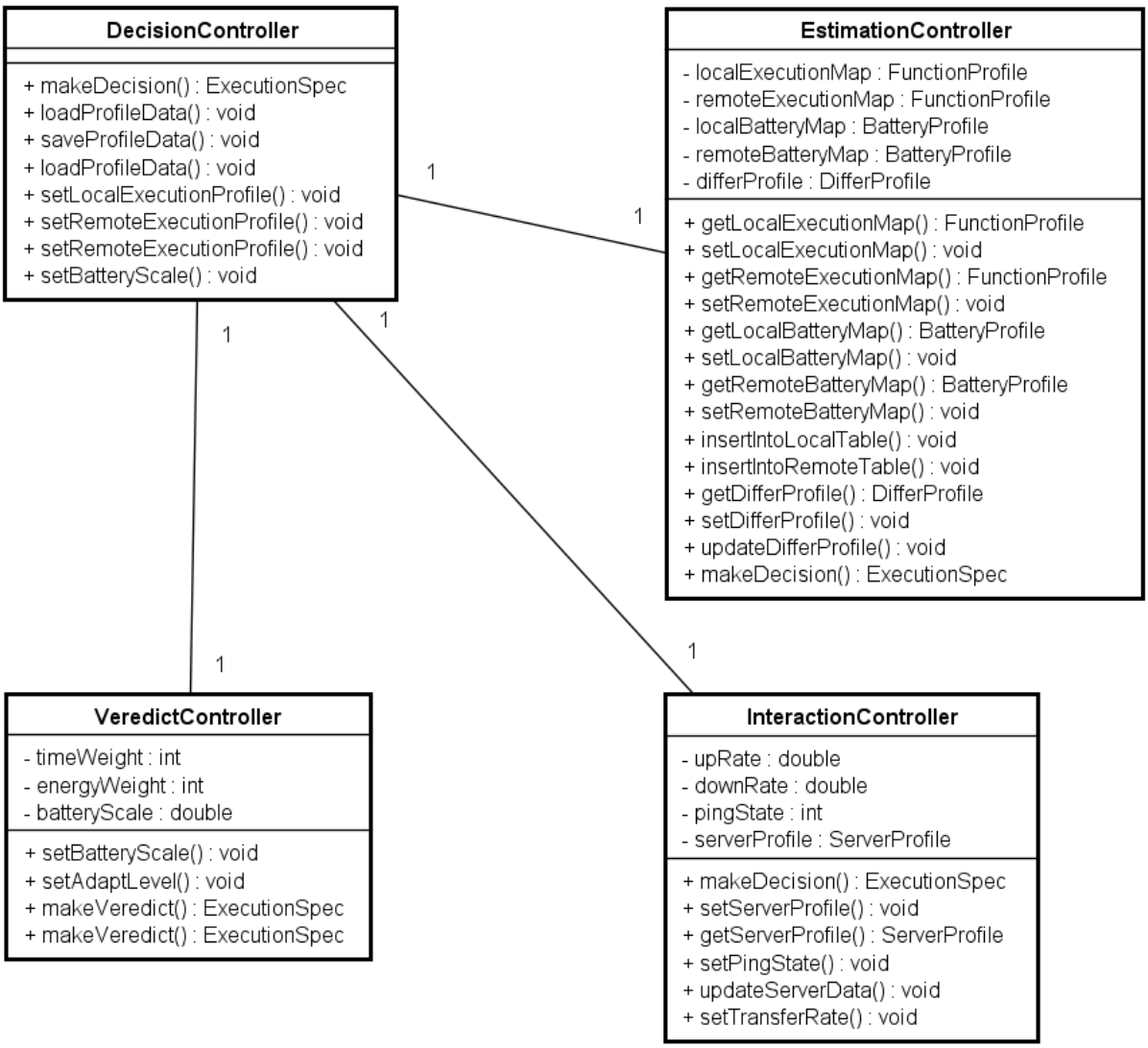
Esta etapa busca estimar o valor aproximado de cada parâmetro de otimização independentemente de dado método com dado parâmetro de entrada. A partir dos valores estimados realiza-se a comparação entre estimação local e remota para decidir se esta chamada será candidata a *offloading* para servidor remoto ou se manterá execução local. Esta comparação é realizada independente para cada parâmetro de otimização.

As principais classes envolvidas nesta etapa são:

- DecisionController, responsável por intermediar a etapa de tomada de decisão de *offloading*;
- EstimationController, responsável pela estimação dos parâmetros de tempo de execução e consumo energético na primeira fase de validação;
- InteractionController, responsável pela estimação dos parâmetros de tempo de execução e consumo energético na segunda fase de validação;
- VeredictController, responsável por elaborar tomada de decisão de *offloading* com base nas estimações realizadas pelas classes EstimationController e InteractionController.

A Figura 4.6 apresenta o relacionamento entre as classes envolvidas nesta etapa.

Figura 4.6: Diagrama de classes para salvar dados de execução utilizados pelo modelo CoSMOS



Elaborado pelo autor.

Para realizar a estimação dos parâmetros adota-se duas fases de estimação, cada uma com abordagens distintas: a primeira fase adota estimação geral do componente, enquanto a segunda fase analisa cada chamada de método isoladamente.

4.3.2 Primeira Fase de Validação

Na primeira fase de validação são realizadas as estimações de tempo de execução e consumo energético para a chamada de um método baseando-se somente no tamanho do parâmetro de entrada. Esta estimação é realizada através de funções matemáticas de primeiro grau, cada uma responsável por estimar um parâmetro de otimização em cada ambiente.

Estimação de Tempo de Execução

O tempo estimado de execução local e remoto para dada chamada de método com determinados parâmetros de entrada são obtidos pelo cálculo das funções matemáticas que seguem o formato da fórmula 4.1 apresentada abaixo:

$$Tempo = e^{(A+B*param)} \tag{4.1}$$

Onde *param* representa as dimensões dos parâmetros de entrada da chamada do método, *Tempo* o valor estimado para o tempo na chamada em questão. A e B são estimadores da fórmula com valores únicos para cada função de estimação, calculados através do uso do método dos mínimos quadrados. O método dos mínimos quadrados é um método de otimização matemática, com o objetivo de estimar valores para a criação de um modelo matemático que melhor se ajuste ao comportamento dos dados observados. Este modelo é obtido através da minimização da soma dos quadrados da diferença entre os valores observados e estimados (FILHO et al., 2011).

Em conjunto com a estimação dos tempos de execução local e remoto, também é utilizada a diferença média ε_{Tempo} entre tempos estimado e observado por cada chamada de método. ε_{Tempo} é calculado pela função matemática 4.2 apresentada abaixo:

$$\varepsilon_{Tempo} = \frac{\sum_{i=1}^{n} |Tempo_{Observado} - Tempo_{Estimado}|}{n} \tag{4.2}$$

A partir dos valores obtidos da estimação de tempo de execução local e remoto pode-se obter decisão de *offloading*:

- Se não há registro do método ter sido executado anteriormente com dada dimensão de parâmetro em ambiente local, este será marcado para **execução local forçado**;
- Se há registro de execução local do método, porém não há para execução remota, este será marcado para ***offloading* forçado**;
- Se o tempo estimado de execução local for maior do que o tempo estimado de execução remota por uma margem ε_{Tempo}, o componente será marcado como **favorável para *offloading***;
- Se o tempo estimado de execução local for menor do que o tempo estimado de execução remota por uma margem $-\varepsilon_{Tempo}$, o componente será marcado como **favorável para execução local**;

- Caso a diferença entre o tempo estimado para execuções local e remota seja inferior a ε_{Tempo}, a validação será marcada como inconclusiva, com tendência para *offloading* caso o tempo estimado local for menor, ou tendência para execução local caso o tempo estimado remoto for menor.

A relação das decisões de *offloading* que podem ser tomadas de acordo com as estimativas de tempo de execução encontram-se resumidas na Tabela 4.2.

Tabela 4.2: Relação de decisões para *offloading* de método de acordo com as estimativas de tempo de execução

Condição das Estimativas	Decisão de *Offloading*
Chamada não executada localmente	Execução local forçada;
Chamada não executada remotamente	*Offloading* forçado;
$Tempo_{Local} - Tempo_{Remoto} > \varepsilon_{Tempo}$	*Offloading* favorável;
$Tempo_{Local} - Tempo_{Remoto} < -\varepsilon_{Tempo}$	Execução local favorável;
$\varepsilon_{Tempo} \geq Tempo_{Local} - Tempo_{Remoto} \geq 0$	*Offloading* inconclusivo. Requer decisão da segunda fase de validação;
$0 > Tempo_{Local} - Tempo_{Remoto} \geq -\varepsilon_{Tempo}$	Execução local inconclusivo. Requer decisão da segunda fase de validação.

Elaborado pelo autor.

Estimação de Consumo Energético

De maneira similar à estimação de tempos de execução, o consumo energético estimado por chamadas local e remota de dado método com parâmetros de entrada com tamanho *param* são obtidos pelas funções matemáticas com formato da fórmula 4.3:

$$Energia = e^{(A+B*param)} \tag{4.3}$$

Onde *param* representa as dimensões dos parâmetros de entrada da chamada do método, *Tempo* o valor estimado para o tempo na chamada em questão. A e B são parâmetros da fórmula com valores únicos para cada função de estimação, também calculados através do uso do método dos mínimos quadrados.

De maneira similar ao tempo de execução, também é utilizada a diferença média $\varepsilon_{Energia}$ entre consumos de energia estimado e observado a partir de cada chamada de método. $\varepsilon_{Energia}$ é calculado pela função matemática 4.4 apresentada abaixo:

$$\varepsilon_{Energia} = \frac{\sum_{i=1}^{n} |Energia_{Observado} - Energia_{Estimado}|}{n} \tag{4.4}$$

A partir dos valores obtidos da estimação de consumo energético local e remoto pode-se obter decisão de *offloading*:

- Se não há registro do método ter sido executado anteriormente com dada dimensão de parâmetro em ambiente local, este será marcado para **execução local forçada**;
- Se há registro de execução local do método, porém não há para execução remota, este será marcado para ***offloading* forçado**;
- Se o consumo estimado de execução local for maior do que o consumo estimado de execução remota por uma margem $\varepsilon_{Energia}$, o componente será marcado como **favorável para *offloading***;
- Se o consumo estimado de execução local for menor do que o consumo estimado de execução remota por uma margem $-\varepsilon_{Energia}$, o componente será marcado como **favorável para execução local**;
- Caso a diferença entre o consumo estimado para execuções local e remota seja inferior a $\varepsilon_{Energia}$, a validação será marcada como inconclusiva, com tendência para *offloading* caso o consumo local estimado for menor, ou tendência para execução local caso o consumo remoto estimado for menor.

A relação das decisões de *offloading* que podem ser tomadas de acordo com as estimativas de consumo energético encontram-se resumidas na Tabela 4.3.

Tabela 4.3: Relação de decisões para *offloading* de método de acordo com as estimativas de consumo energético

Condição das Estimativas	Decisão de *Offloading*
Chamada não executada localmente	Execução local forçada;
Chamada não executada remotamente	*Offloading* forçado;
$Energia_{Local} - Energia_{Remoto} > \varepsilon_{Energia}$	*Offloading* favorável;
$Energia_{Local} - Energia_{Remoto} < -\varepsilon_{Energia}$	Execução local favorável;
$\varepsilon_{Energia} \geq Energia_{Local} - Energia_{Remoto} \geq 0$	*Offloading* inconclusivo. Requer decisão da segunda fase de validação;
$0 > Energia_{Local} - Energia_{Remoto} \geq -\varepsilon_{Energia}$	Execução local inconclusivo. Requer decisão da segunda fase de validação.

Elaborado pelo autor.

Tomada de Decisão

Para tomar a decisão final é necessário levar em consideração as decisões já obtidas a partir dos parâmetros de tempo de execução e consumo energético. Porém estas decisões podem ser conflitantes entre si, desta forma dificultando a obtenção de uma decisão que atenda a ambos os objetivos de otimização.

Para solucionar este problema resolveu-se adotar a seguinte abordagem: adotar um valor numérico para cada tipo de decisão, e somar os valores obtidos de cada decisão de parâmetro de acordo com a seguinte fórmula matemática:

$$Decisao_{Geral} = Decisao_{Tempo} + Decisao_{Energia} \tag{4.5}$$

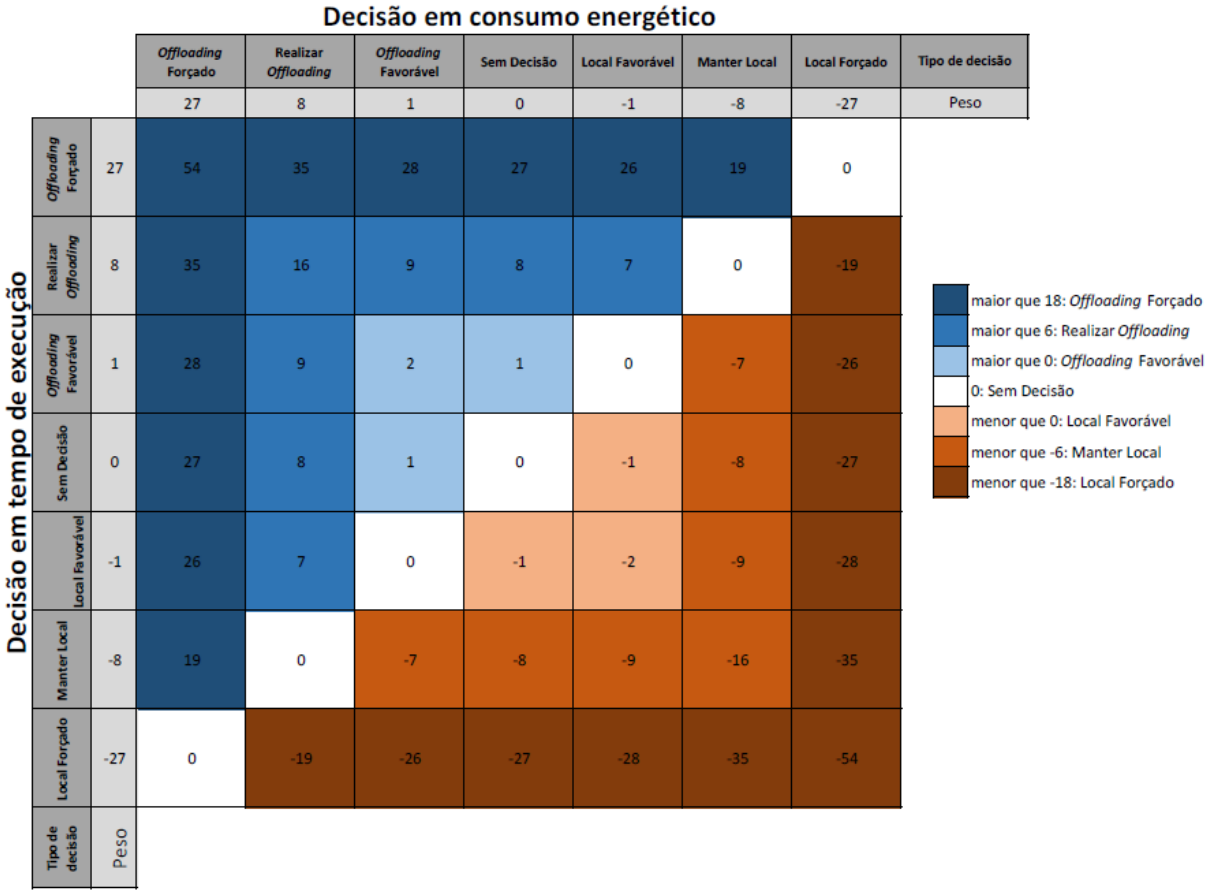
Cada um dos tipos de decisão de ambos os parâmetros de otimização possuem os seguintes valores:

- 27: *Offloading* forçado;
- 8: *Offloading* favorável;
- 1: *Offloading* inconclusivo;
- 0: Sem decisão;
- -1: Local inconclusivo;
- -8: Local favorável;

- -27: Local forçado.

A relação entre as decisões dos parâmetros de tempo de execução e consumo energético encontra-se ilustrado no diagrama da figura 4.7.

Figura 4.7: Diagrama de tomada de decisão em relação a tempo de execução e consumo energético



Elaborado pelo autor.

A relação entre as decisões pelos dois parâmetros de otimização foi modelada de acordo com as seguintes premissas:

- Se ambos os parâmetros possuírem mesma decisão, seguir com a decisão acordada;
- Se um dos valores de execução local não foi encontrado, então forçar execução local;
- Se um dos valores de execução remota não foi encontrado, então forçar execução remota;
- Se somente um dos parâmetros decidir por *offloading*, com o outro somente for favorável a alguma das decisões, então opta-se for *offloading*;

- Similarmente, se somente um dos parâmetros decidir por execução local, com o outro somente for favorável a alguma das decisões, então opta-se for manter execução local;
- Se um dos parâmetros optar por favorecer *offloading*, mas o outro não possuir decisão formada, então optar por *offloading* favorável;
- Similarmente, se um dos parâmetros optar por favorecer execução local, mas o outro não possuir decisão formada, então optar por execução local favorável;
- Para os demais casos será considerado como nenhuma decisão tomada, que será definida pela segunda fase de validação.

4.3.3 Segunda Fase de Validação

Na segunda fase de validação são realizadas as estimações mais precisas de tempo de execução e consumo energético para a chamada de um método com dado tamanho de parâmetro de entrada com base no histórico de execução.

Esta fase somente será realizada se a decisão tomada pela primeira fase for de execução local favorável, *offloading* favorável, ou nenhuma decisão tomada.

A estimação dos parâmetros de otimização é realizada através de funções matemáticas específicas para cada parâmetro de otimização.

Estimação de Tempo de Execução

Para a estimação de tempo de execução de dada chamada a método com determinados parâmetros de entrada são empregadas funções matemáticas. Porém, neste caso são empregadas funções de formatos distintos para a estimação de execução local e remota, por estas serem influenciadas por fatores distintos (AHMED et al., 2015; VERBELEN et al., 2012; KOVACHEV; YU; KLAMMA, 2012).

Para estimação de tempo de execução local adota-se o tempo médio observado, seguindo a fórmula 4.6 abaixo.

$$Tempo_{Local} = \frac{\sum_{i=1}^{n} Execucao_{Obs}}{n} \tag{4.6}$$

Onde $Execucao_{Obs}$ representa o tempo de execução observado para dada chamada, n o número de chamadas registradas no histórico de execução, e $Tempo_{Local}$ o valor médio do tempo de execução.

Para estimação de tempo de execução remota adota-se o tempo médio observado acrescido do tempo estimado de tráfego de dados entre dispositivo e servidor remoto, seguindo a fórmula 4.7 abaixo.

$$Tempo_{Remota} = \frac{\sum_{i=1}^{n} Execucao_{Obs}}{n} + \frac{Parametro}{Taxa_{Upload}} + \frac{Retorno}{Taxa_{Download}} + RTT \tag{4.7}$$

Onde $Execucao_{Obs}$ representa o tempo de execução observado para dada chamada, n o número de chamadas registradas no histórico de execução, *Parametro* as dimensões do parâmetro de entrada da chamada ao componente, $Taxa_{Upload}$ a taxa de transferência de dados do dispositivo para servidor, *Retorno* as dimensões do parâmetro de retorno da chamada ao componente, $Taxa_{Download}$ a taxa de transferência de dados do servidor para o dispositivo, e *RTT* o tempo médio de estabelecimento de comunicação entre dispositivo e servidor.

A partir dos valores obtidos das funções de tempo médio de execução local e remoto estimado pode-se obter a decisão de *offloading* dentre duas opções:

- Se o tempo estimado de execução local for maior do que o tempo estimado de execução remota, a funcionalidade será marcada como **favorável para *offloading***;
- Se o tempo estimado de execução local for menor do que o tempo estimado de execução remota, a funcionalidade será marcada como **favorável para execução local**;

Estimação de Consumo de Energia

Similarmente à abordagem adotada para estimação de tempo, adota-se funções matemáticas para estimação de consumo energético para cada chamada de método com determinadas dimensões de parâmetro de entrada. Entretanto, neste caso adota-se funções com formato similar entre si para a estimação, baseando-se no consumo médio observado a partir das informações de histórico de execução para cada ambiente execução. Estas funções são representadas pela fórmula 4.8 abaixo:

$$Energia_{Int} = \frac{\sum_{i=1}^{n} Energia_{Obs}}{n} \tag{4.8}$$

Onde $Energia_{Obs}$ representa o consumo energético observado, n o número de execuções registrados, e $Energia_{Int}$ o valor médio do consumo energético para dada chamada de método.

A partir dos valores obtidos das funções de consumo energético médio local e remoto pode-se obter decisão de *offloading* dentre duas possibilidades:

- Se o consumo energético médio local for maior do que o consumo energético médio remoto, o método será marcado como **favorável para *offloading***;
- Se o consumo energético médio local for menor do que o consumo energético médio remoto, o método será marcado como **favorável para execução local**;

Tomada de Decisão A tomada de decisão através das estimações de tempo de execução e consumo energético segue a seguinte heurística:

- Caso ambas as estimações sejam favoráveis à execução remota, então o método será marcada para **execução remota**;
- Caso ambas as estimações sejam favoráveis à execução local, então o método será marcada para **execução local**;
- Para os demais casos, o método será marcada como decisão não tomada.

Para os casos em que a decisão não foi tomada, será levada em conta a decisão mais favorável obtida na primeira fase. E se não houve decisão tomada em ambas as fases, o método será considerado para *offloading*.

4.3.4 Etapa de Execução do Método

Após elaboração da tomada de decisão de *offloading* computacional se dará a subetapa de execução do método, realizado pelo MpOS. Nesta subetapa o *framework* estabelecerá conexão com servidor remoto caso opte por realizar execução remota, ou prosseguirá para execução local caso contrário.

4.3.5 Etapa de Coleta de Dados

A subetapa de coleta de dados é realizada logo após o término da execução de cada uma das chamadas de método, independentemente da chamada ter sido realizada local ou remotamente.

Após realizada a execução do método invocado, MpOS chamará o método setLocalExecutionProfile() da classe DecisionController para caso a execução tenha sido realizada localmente, ou setRemoteExecutionProfile() caso tenha sido realizada remotamente.

Para ambos os métodos serão passadas informações de tamanho do parâmetro de entrada, tempo de execução total da funcionalidade, consumo energético do dispositivo durante sua execução e estimativas de tempo de execução e consumo energético.

Para a coleta de informações sobre tempo de execução mede-se somente a diferença de tempo entre antes e depois da execução do método invocado. Já para a coleta de informações sobre consumo energético utiliza-se a biblioteca BatteryManager da própria API do Android, coletando informações sobre corrente elétrica e tensão da bateria do dispositivo antes e depois a execução do método.

Para obter a potência média da bateria a partir dos valores de corrente e tensão utiliza-se a seguinte equação 4.9:

$$Potencia(Watt) = Tensao(Volt) * Corrente(Ampere) \tag{4.9}$$

Com o valor de potência obtido por 4.9 calcula-se o valor do consumo energético aproximado pela execução do método através da seguinte equação 4.10:

$$Energia(Joule) = Potencia(Watt) * Tempo(segundo) \tag{4.10}$$

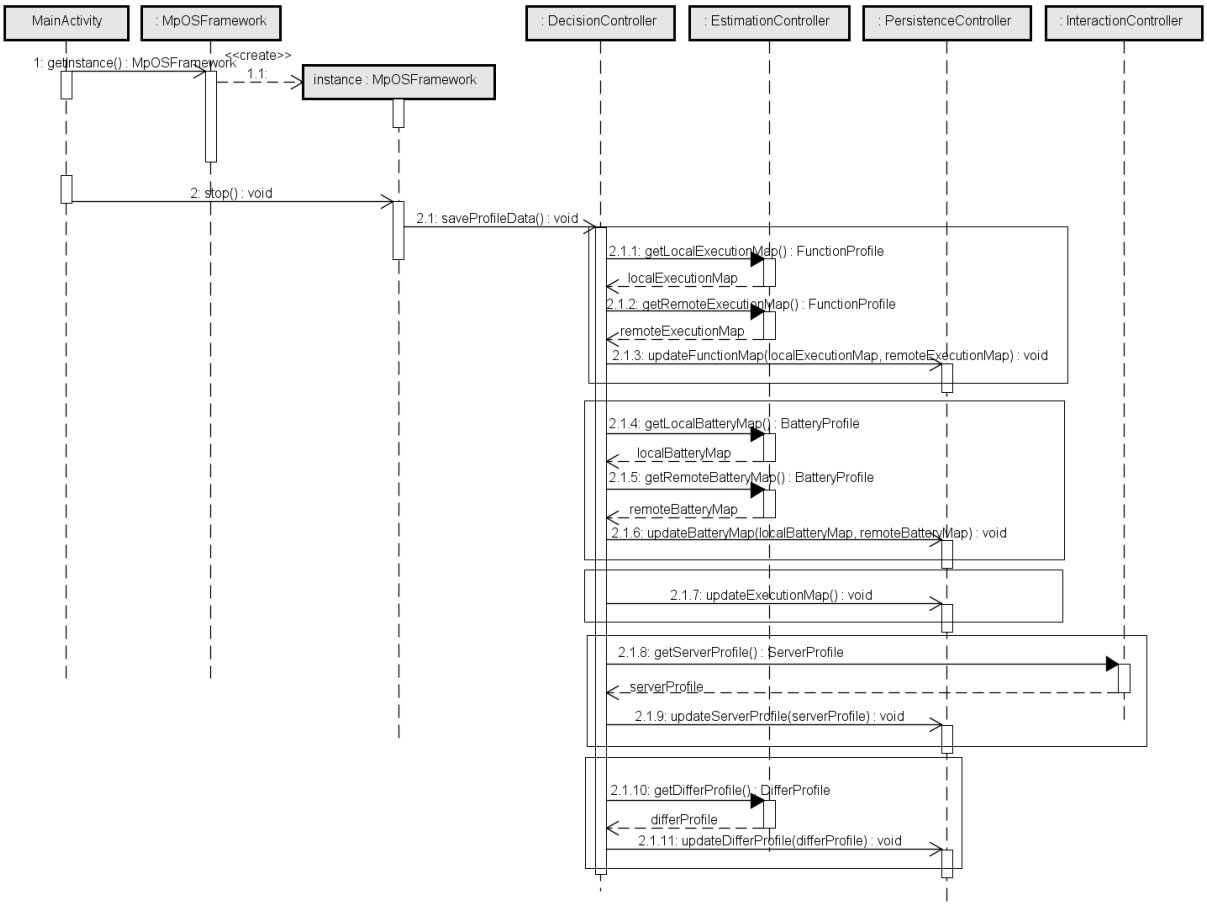
Através da equação 4.10 obtém-se o valor aproximado de consumo energético através dos valores de potência e do tempo de execução, este já obtido anteriormente.

4.4 Persistência de Informações

Por fim, a etapa de persistência é realizada durante a fase de parada da aplicação, juntamente com o encerramento do MpOS. É nesta etapa onde todas as informações utilizadas para estimação dos parâmetros de otimização serão armazenados na base de dados SQLite do próprio *framework*.

A Figura 4.8 apresenta a seqüência de operações adotada por CoSMOS para salvar no banco de dados SQLite os dados relativos às informações de execuções da aplicação móvel, representando através de diagrama de sequência UML.

Figura 4.8: Diagrama de sequência para salvar dados de execução utilizados pelo modelo CoSMOS



Elaborado pelo autor.

Ao encerrar a aplicação móvel, a classe MainActivity invoca o método stop() do MpOS, que iniciará a liberação de recursos utilizados pelos componentes do *framework*. Antes de encerrar todos os seus componentes, o próprio MpOS invocará o método saveProfileData() do componente DecisionController, responsável por persistir todas as informações utilizadas para estimação de execução das funcionalidades e tomada de decisão.

A classe PersistenceController é responsável por persistir todos os dados na base SQLite, realizando as operações através dos métodos updateFunctionMap(), updateBatteryMap(), updateExecutionMap(), updateServerData() e updateDifferProfile().

4.5 Considerações Finais do Capítulo

Este capítulo apresentou a implementação do modelo CoSMOS, descrição geral das principais classes utilizadas e principais fluxos de operação utlizados.

Devido ao fato do MpOS possuir implementado acesso à base de dados SQLite para persistência de dados, não houve necessidade de desenvolver novo meio de acesso, sendo suficiente utilizar a classe base Dao para este fim.

No próximo capítulo será apresentada a avaliação do modelo CoSMOS realizada através de dois estudos de casos.

Capítulo 5

ESTUDOS DE CASO

Este capítulo tem como objetivo apresentar os experimentos conduzidos para avaliar o modelo CoSMOS sobre quatro ambientes distintos de execução por meio de uso em duas aplicações móveis. Também foi conduzido em conjunto avaliação da acurácia de estimativas realizadas pela primeira fase de validação através destas aplicações.

5.1 Objetivos dos Experimentos

Por meio de dois estudos de caso, esta etapa do trabalho possui três objetivos principais para avaliação.

Primeiramente, esta busca analisar a capacidade do modelo de adaptar a execução de aplicações móveis para obter melhor configuração de execução de acordo com as condições do ambiente onde se encontra.

Juntamente com a análise da capacidade de adaptação, esta etapa busca analisar as decisões tomadas para *offloading* dos componentes conforme as condições de cada ambiente, além de como são utilizadas as abordagens distintas através dos dois estágios de validação.

Também é avaliada a acurácia da estimação de parâmetros de otimização para as fórmulas matemáticas empregadas na primeira fase de validação, com base nos resultados obtidos pelos estudos de caso.

5.2 Base dos Experimentos

Para avaliar foram utilizadas duas aplicações como prova de conceito, sendo que uma destas foi especialmente desenvolvida para realizar a avaliação. A outra aplicação foi desenvolvida originalmente como prova de conceito para a plataforma MpOS (COSTA et al., 2015).

Enquanto a aplicação das N rainhas avalia carga de processamento para dados de tamanho reduzido, a aplicação BenchImage avalia carga de processamento e taxa de transmissão de dados para arquivos de imagens como parâmetro de entrada, sendo que estes arquivos possuem tamanho significantemente grandes.

Estas aplicações foram executadas em quatro ambientes distintos:

1. Execução exclusivamente local;
2. Execução remota através de WiFi para um servidor *cloudlet*, um servidor remoto que encontra-se dentro da mesma rede que o dispositivo;
3. Execuções remotas através de WiFi para um servidor *cloud* hospedada na nuvem pública da Amazon;
4. Execuções remotas através de 3G para um servidor *cloud* hospedada na nuvem pública da Amazon.

As especificações técnicas tanto do dispositivo móvel utilizado como dos servidores remotos encontram-se dispostos na Tabela 5.1

Tabela 5.1: Especificações e dados técnicos das ferramentas utilizadas nos experimentos

Ferramenta	**Dados Técnicos**
Dispositivo Móvel	Motorola Moto G 1a Geração - Android 5.0 Lollipop Processador Qualcomm Snapdragon quad-core, 1.2GHz e 1GB de RAM.
Plataforma *Cloudlet*	Windows 7 Profissional SP1 64-bits Processador Intel Core i5-3230M, dual core, 2.6GHz e 6GB de RAM.
Plataforma *Cloud*	Amazon Linux AMI 2017.03.1 (HVM) / t2.micro / São Paulo-SP 1 vCPU, Intel Xeon 2.5GHz

Elaborado pelo autor.

Para medição de tempo de execução em cada chamada de funcionalidade, cada aplicação disponibiliza o tempo de execução na tela do dispositivo. Para medição de consumo energético adotou-se monitoramento por meio da aplicação Trepn[1], por este ser uma das aplicações de monitoramento de energia que oferece melhor precisão nas informações coletadas (HOQUE et al., 2015).

5.3 Avaliação de Funções de Estimativa

O objetivo desta avaliação é medir a acurácia de duas funções matemáticas em aproximar valores estimados para cada chamada de método com entrada em dadas faixas de dimensão, comparando aos valores médios observados da execução prática de cada chamada. A partir dessas comparações busca-se escolher aquela que obtiver melhor aproximação para uso prático no modelo.

5.3.1 Estrutura da Avaliação

Para avaliar a acurácia das fórmulas candidatas à estimação de parâmetros de otimização foram utilizados as duas aplicações móveis: *BenchImage* e a implementação do problema das N rainhas, com base nas informações de execução relativas àquelas realizadas exclusivamente dentro do dispositivo móvel.

Foram escolhidas duas funções matemáticas para estimação de parâmetros:

- Função linear, da forma $\hat{Y} = A + BX$;
- Função exponencial, da forma $\hat{Y} = e^{A+BX}$.

Onde $\hat{Y}$ é o valor estimado do parâmetro de otimização, X a dimensão do parâmetro de entrada do componente, A e B são estimadores exclusivos de cada função matemática, a serem calculados através do uso do método dos mínimos quadrados.

Devido às limitações de representação através deste método, foram empregadas somente funções com dois estimadores, seguindo os formatos apresentados.

Ambas as funções candidatas foram submetidas às duas aplicações, avaliando sua acurácia de estimação para os dois parâmetros de otimização: tempo de execução e consumo energético.

[1] https://developer.qualcomm.com/software/trepn-power-profiler

5.3.2 Avaliação com a Aplicação *BenchImage*

A avaliação realizada através da aplicação *BenchImage* foi dividida entre comparações em tempo de execução e em consumo energético.

Tempo de Execução

Com base nos estimadores calculados a partir do histórico de execução da aplicação para execução local, as funções linear e exponencial obtiveram as formas apresentadas respectivamente pelas fórmulas 5.1 e 5.2.

$$\hat{y}_{linear} = 0,212 + 17,910x \quad (5.1)$$

$$\hat{y}_{exp} = e^{(1,100+1,256x)} \quad (5.2)$$

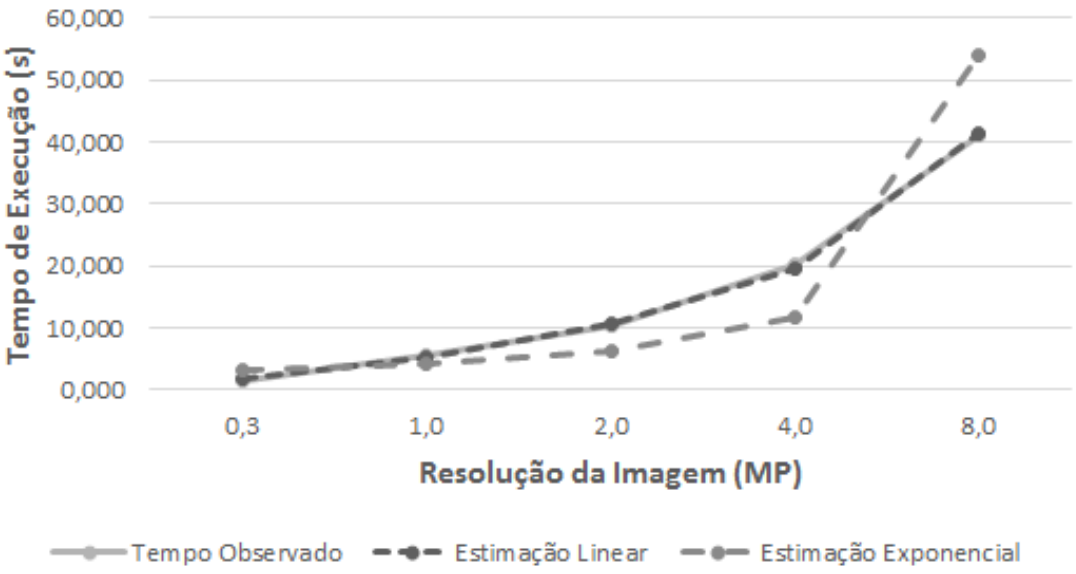
Os valores médios de tempo de execução obtidos de observação da execução local prática e os valores estimados para tempo de execução a partir das funções matemáticas 5.1 e 5.2 encontram-se dispostos na Tabela 5.2 e representados graficamente na Figura 5.1.

Tabela 5.2: Comparação entre modelos de estimação linear e exponencial para tempo de execução e tempo médio observado para a aplicação BenchImage em segundos

Resolução	Modelo Linear	Modelo Exponencial	Tempo observado
0.3	1,880	3,377	1,654
1.0	5,353	4,308	5,491
2.0	10,801	6,312	10,472
4.0	19,530	11,639	20,172
8.0	41,370	53,811	41,146

Elaborado pelo autor.

Figura 5.1: Gráfico de comparação entre modelos de estimação linear e exponencial para tempo de execução e tempo médio observado para a aplicação *BenchImage*



Elaborado pelo autor.

Consumo Energético

Com base nos estimadores calculados a partir do histórico de execução local da aplicação *BenchImage*, as funções linear e exponencial obtiveram as formas apresentadas respectivamente pelas fórmulas 5.3 e 5.4.

$$\hat{y}_{linear} = -0,361 + 12,515x \tag{5.3}$$

$$\hat{y}_{exp} = e^{(0,564+1,336x)} \tag{5.4}$$

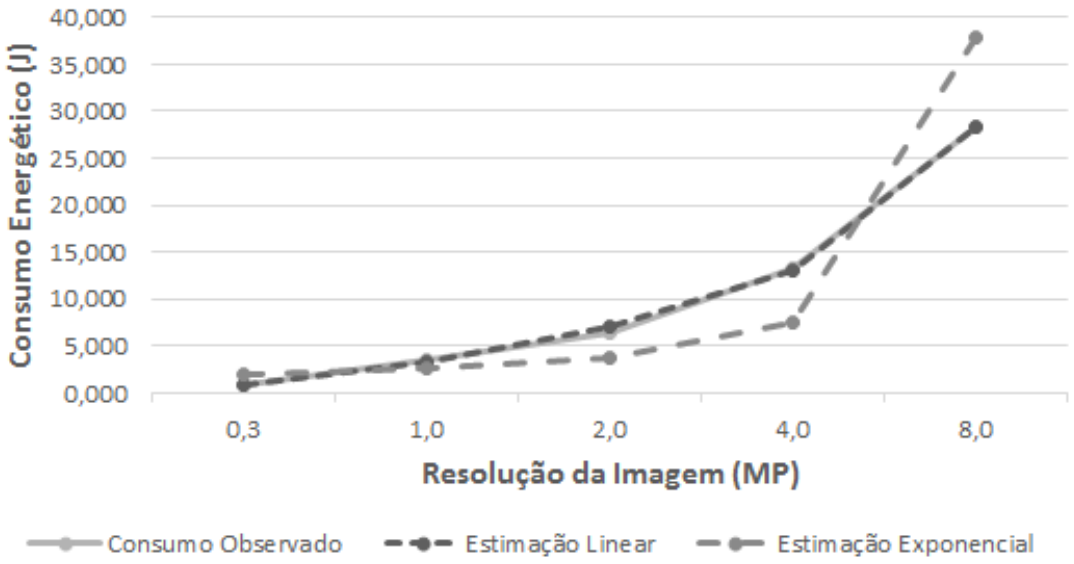
Os valores médios de consumo energético obtidos de observação da execução local prática e os valores estimados para consumo energético a partir das funções matemáticas 5.3 e 5.4 encontram-se dispostos na Tabela 5.3 e representados graficamente na Figura 5.2.

Tabela 5.3: Comparação entre modelos de estimação linear e exponencial para consumo energético e consumo médio observado para a aplicação BenchImage em Joules

Resolução	Modelo Linear	Modelo Exponencial	Consumo observado
0.3	0,804	1,991	0,926
1.0	3,231	2,579	3,451
2.0	7,038	3,872	6,485
4.0	13,138	7,425	13,329
8.0	28,400	37,845	28,420

Elaborado pelo autor.

Figura 5.2: Gráfico de comparação entre modelos de estimação linear e exponencial para consumo energético e consumo médio observado para a aplicação *BenchImage*



Elaborado pelo autor.

Baseando-se nos resultados obtidos para tempo de execução e consumo energético, a estimação através de função linear obteve melhor aproximação para os valores observados do que a função exponencial para ambos os parâmetros de otimização na aplicação *BenchImage*.

5.3.3 Avaliação com a Aplicação das N Rainhas

A avaliação realizada através da aplicação das N rainhas foi dividida entre comparações em tempo de execução e em consumo energético, da mesma forma como foi realizado com a avaliação pelo *BenchImage*.

Tempo de Execução

Com base nos parâmetros das funções, calculados a partir do histórico de execução da aplicação, as funções linear e exponencial obtiveram as formas apresentadas respectivamente pelas fórmulas 5.5 e 5.6.

$$\hat{y}_{linear} = -201,71667 + 829393,7813x \quad (5.5)$$

$$\hat{y}_{exp} = e^{(-16,18586+59598,4393x)} \quad (5.6)$$

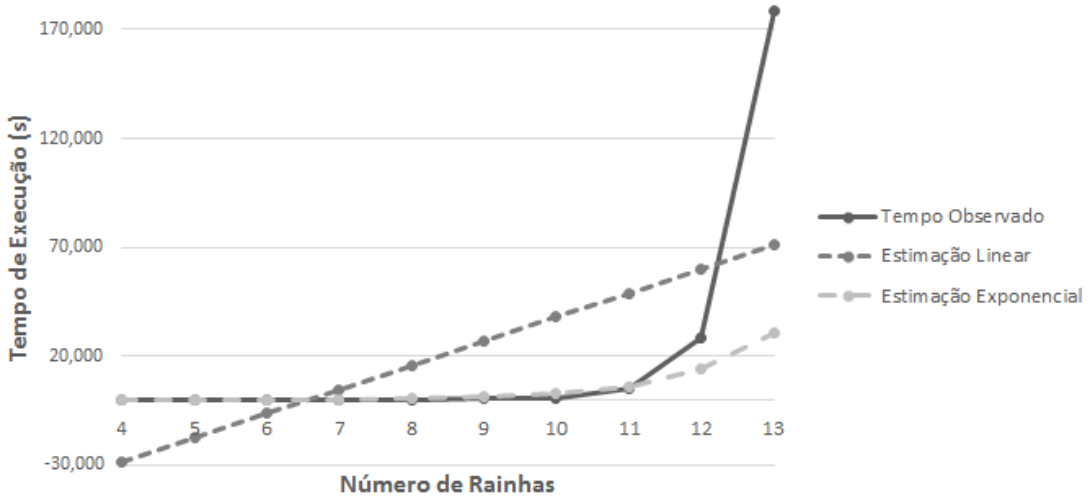
Os valores médios de tempo de execução obtidos de observação da execução local prática e os valores estimados para tempo de execução a partir das funções matemáticas 5.5 e 5.6 encontram-se dispostos na Tabela 5.4 e representados graficamente na Figura 5.3.

Tabela 5.4: Comparação entre modelos de estimação linear e exponencial para tempo de execução e tempo médio observado para a aplicação das N rainhas em segundos

Nro de rainhas	Modelo Linear	Modelo Exponencial	Tempo observado
4	-28,494	0,024	0,113
5	-17,420	0,053	0,111
6	-6,347	0,117	0,121
7	4,727	0,259	0,124
8	15,800	0,574	0,165
9	26,872	1,271	0,317
10	37,945	2,817	1,010
11	49,021	6,244	4,845
12	60,098	13,840	28,283
13	71,169	30,644	178,282

Elaborado pelo autor.

Figura 5.3: Gráfico de comparação entre modelos de estimação linear e exponencial para tempo de execução e tempo médio observado para a aplicação das N rainhas



Elaborado pelo autor.

Consumo Energético

Com base nos parâmetros das funções, calculados a partir do histórico de execução da aplicação, as funções linear e exponencial obtiveram as formas apresentadas respectivamente pelas fórmulas 5.7 e 5.8.

$$\hat{y}_{linear} = -119,7024 + 491850,682x \tag{5.7}$$

$$\hat{y}_{exp} = e^{(-17,19917+60798,09969x)} \tag{5.8}$$

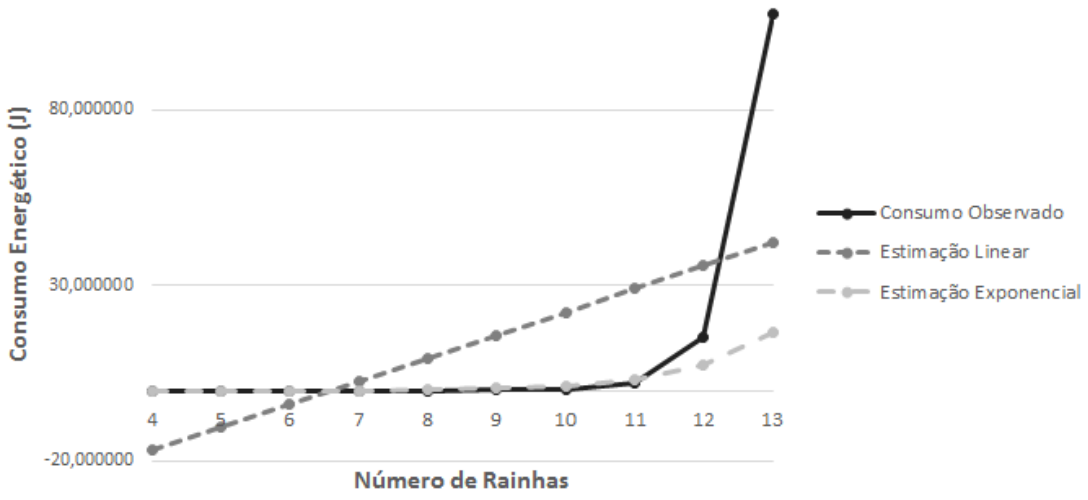
Os valores médios de consumo energético obtidos de observação da execução local prática e os valores estimados para consumo energético a partir das funções matemáticas 5.7 e 5.8 encontram-se dispostos na Tabela 5.5 e representados graficamente na Figura 5.4.

Tabela 5.5: Comparação entre modelos de estimação linear e exponencial para consumo energético e consumo médio observado para a aplicação das N rainhas em Joules

Nro de rainhas	Modelo Linear	Modelo Exponencial	Consumo observado
4	-16,977	0,011	0,054
5	-10,410	0,025	0,053
6	-3,843	0,056	0,062
7	2,724	0,127	0,063
8	9,290	0,285	0,084
9	15,857	0,642	0,156
10	22,423	1,446	0,504
11	28,991	3,258	2,275
12	35,560	7,337	15,236
13	42,125	16,519	107,253

Elaborado pelo autor.

Figura 5.4: Gráfico de comparação entre modelos de estimação linear e exponencial para consumo energético e consumo médio observado para a aplicação das N rainhas



Elaborado pelo autor.

Baseando-se nos resultados obtidos para tempo de execução e consumo energético, a estimação através de função exponencial obteve melhor aproximação para os valores observados do que a função linear para ambos os parâmetros de otimização na aplicação das N rainhas.

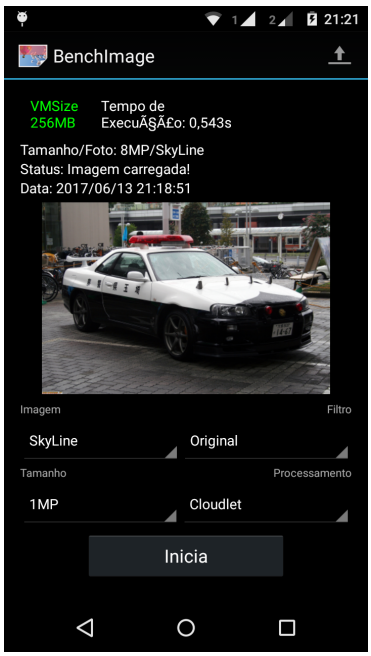
5.3.4 Discussão dos Resultados

Embora seja possível obter comportamento linear aproximado através de função exponencial, não é possível obter comportamento exponencial aproximado através de função linear. Portanto optou-se por utilizar estimação de parâmetros através de função exponencial.

5.4 Avaliação da Aplicação *BenchImage*

BenchImage é uma aplicação móvel de processamento de imagens desenvolvida por Costa et al. (2015) com a finalidade de aplicar filtro sobre uma imagem selecionada pelo usuário. A tela inicial da aplicação encontra-se ilustrada na Figura 5.5.

Figura 5.5: Tela inicial da aplicação BenchImage



Baseado em Costa et al. (2015).

A aplicação disponibiliza quatro filtros distintos a serem aplicados sobre três imagens, cada qual possuindo cinco variações de resolução entre 0,3 MP e 8,0 MP. O uso dos quatro filtros encontra-se ilustrado na Figura 5.6.

Como ambas as plataformas de suporte MpOS e CoSMOS acomplam-se de forma idêntica à aplicação móvel, não houve necessidade de alteração estrutural na aplicação original.

Figura 5.6: Exemplos de aplicação de filtros do BenchImage sobre uma foto: Original, Cartoonizer, Red Tone e Sharpen

Baseado em Costa et al. (2015).

5.4.1 Cenários de Execução

As execuções da aplicação *BenchImage* foram divididas em oito séries, cada qual constituída por dez ciclos, com cada ciclo formado por uma imagem em cinco resoluções distintas entre 0.3 MP e 8.0 MP e submetida a três filtros distintos. Todas as séries foram distribuídas igualmente pelos quatro ambientes de estudo: execução exclusivamente no dispositivo móvel, execução sob apoio por servidor *Cloudlet* com conexão via WiFi, apoio por *Cloud* com conexão via WiFi, e apoio por *Cloud* com conexão via 3G.

5.4.2 Resultados Obtidos

A seguir serão apresentados os resultados obtidos e análise da série de execuções para cada número de rainha sob duas perspectivas: tempo médio de execução e consumo energético médio.

Tempo de Execução

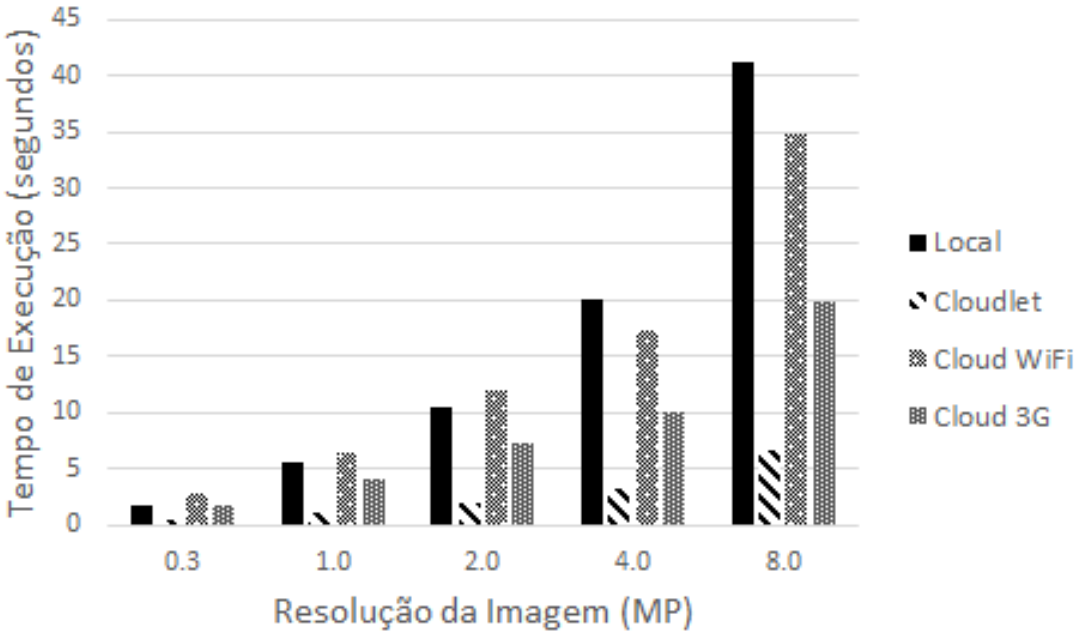
A Tabela 5.6 e o gráfico da Figura 5.7 apresentam informações sobre o tempo médio e desvio padrão de execução da aplicação *BenchImage* para a imagem *FAB Show* em cada uma das cinco resoluções disponíveis. O tempo de execução de cada avaliação foi obtido através de mensuramento pela própria aplicação.

Tabela 5.6: Tempo médio em segundos de execução do BenchImage por resolução de imagem

Resolução	**Suporte**			
	Local	***Cloudlet* (WiFi)**	***Cloud* (WiFi)**	***Cloud* (3G)**
0.3	1,654 ± 0,546	0,455 ± 0,150	2,681 ± 2,315	1,702 ± 0,656
1.0	5,491 ± 1,930	1,010 ± 0,396	6,408 ± 2,970	4,110 ± 1,228
2.0	10,472 ± 3,688	1,864 ± 0,903	12,043 ± 6,223	7,345 ± 2,658
4.0	20,172 ± 7,079	3,195 ± 1,989	17,275 ± 3,828	9,995 ± 3,414
8.0	41,146 ± 14,136	6,723 ± 3,008	34,805 ± 8,421	19,765 ± 10,423

Elaborado pelo autor.

Figura 5.7: Gráfico de tempo de execução para a aplicação BenchImage por resolução de imagem



Elaborado pelo autor.

A Tabela 5.7 apresenta informações sobre tempo médio geral e desvio padrão de execução da aplicação *BenchImage* para cada um dos quatro ambientes de execução.

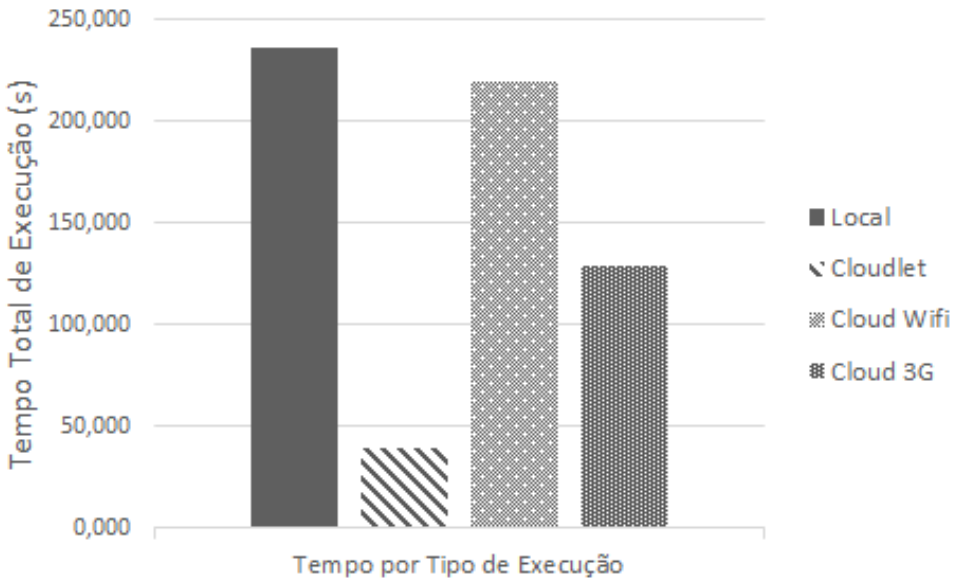
Tabela 5.7: Comparação de tempo médio e desvio padrão geral de execução em segundos da aplicação BenchImage por ciclo de avaliação em cada ambiente

Suporte	Tempo de Execução Total Médio por Ciclo de Avaliação
Local no Dispositivo	236,803 ± 15,945
Plataforma *Cloudlet* (WiFi)	39,741 ± 8,169
Plataforma *Cloud* (WiFi)	219,638 ± 13,105
Plataforma *Cloud* (3G)	128,753 ± 16,334

Elaborado pelo autor.

Os resultados da Tabela 5.7 relativos ao tempo de execução médio geral encontra-se ilustrado pelo gráfico na Figura 5.8.

Figura 5.8: Gráfico de tempo de execução para a aplicação BenchImage por ciclo de avaliação



Elaborado pelo autor.

Com base nos resultados obtidos, nota-se que há tanto diferenças como similaridades entre as performances de tempo de execução dentro dos quatro ambientes de execução. Enquanto observou-se performance semelhante para execução local e uso de suporte *cloud* para resolução de imagens entre 0.3 e 2.0 MP, para resoluções superiores a execução local demonstrou performance pior que os demais.

Comparado aos demais, o suporte por servidor *cloudlet* demonstrou melhor performance em todas as resoluções de imagem utilizadas.

Essa diferença de performance em relação à execução local se deve ao uso do *offloading* computacional para servidores remotos, que busca diminuir o tempo de execução ao empregar uso de recursos externos ao dispositivo móvel para este fim.

Consumo Energético

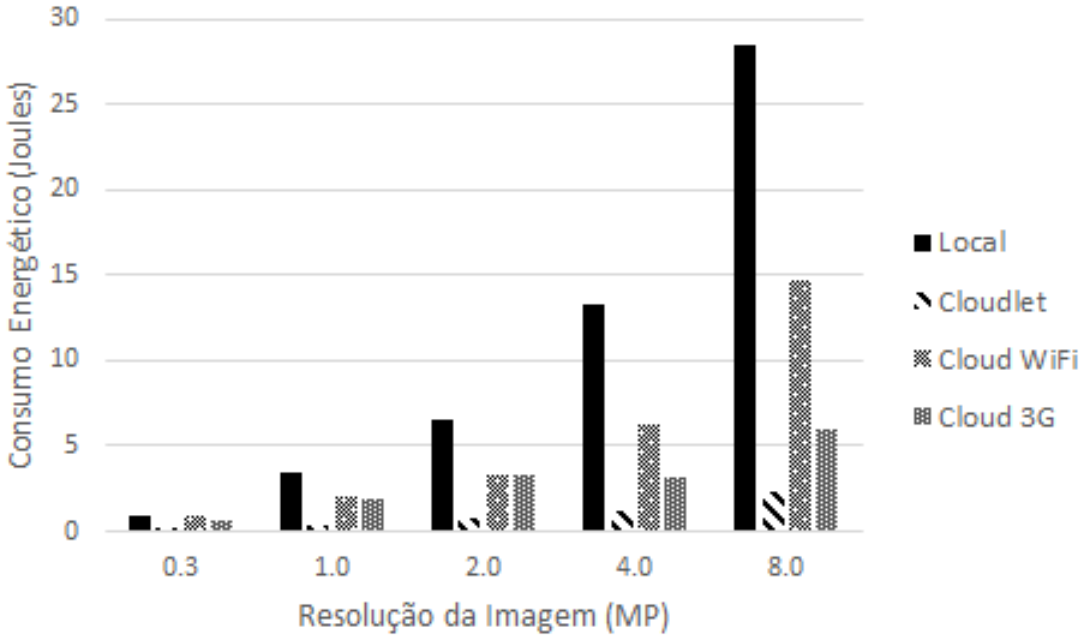
A Tabela 5.8 e o gráfico da Figura 5.9 apresentam informações sobre o consumo energético médio e desvio padrão durante execução da aplicação *BenchImage* para a imagem *FAB Show* em cada uma das cinco resoluções. O consumo energético foi mensurado através da aplicação Trepn.

Tabela 5.8: Consumo energético médio em joules pela execução do BenchImage por resolução de imagem

Resolução	**Suporte**			
	Local	***Cloudlet* (WiFi)**	***Cloud* (WiFi)**	***Cloud* (3G)**
0.3	0,926 ± 0,339	0,192 ± 0,059	0,891 ± 0,568	0,663 ± 0,250
1.0	3,451 ± 1,206	0,416 ± 0,129	2,055 ± 0,569	1,831 ± 0,772
2.0	6,485 ± 2,236	0,771 ± 0,284	3,296 ± 1,185	3,316 ± 1,698
4.0	13,329 ± 4,436	1,217 ± 0,743	6,222 ± 2,472	3,101 ± 1,071
8.0	28,420 ± 10,192	2,361 ± 0,944	14,686 ± 6,733	5,910 ± 5,567

Elaborado pelo autor.

Figura 5.9: Gráfico de consumo energético para a aplicação BenchImage por resolução de imagem



Elaborado pelo autor.

A Tabela 5.9 apresenta informações sobre o consumo energético médio e desvio padrão geral da aplicação *BenchImage* para cada um dos quatro ambientes de execução.

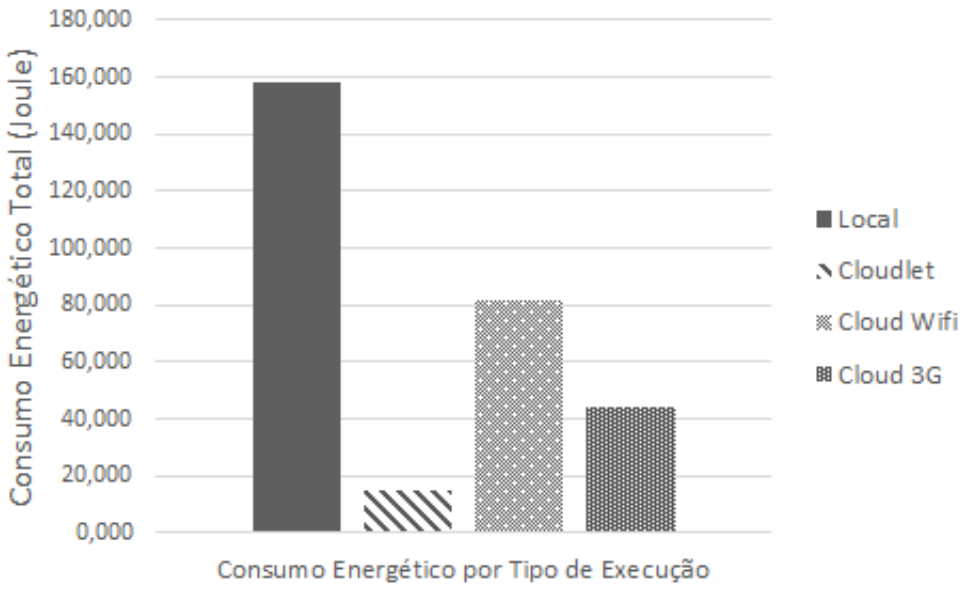
Tabela 5.9: Comparação de consumo energético médio e desvio padrão geral em joules por execução de ciclo de avaliação da aplicação BenchImage em cada ambiente

Suporte	Consumo Energético Total Médio por Ciclo de Avaliação
Local no Dispositivo	157,833 $\pm$ 11,304
Plataforma *Cloudlet* (WiFi)	14,872 $\pm$ 4,723
Plataforma *Cloud* (WiFi)	81,451 $\pm$ 6,770
Plataforma *Cloud* (3G)	44,464 $\pm$ 4,831

Elaborado pelo autor.

Os resultados da Tabela 5.9 relativos ao consumo energético médio geral encontra-se ilustrado pelo gráfico na Figura 5.10.

Figura 5.10: Gráfico de consumo energético para a aplicação BenchImage por ciclo de avaliação



Elaborado pelo autor.

Com base nos resultados obtidos, nota-se que o desempenho para consumo energético foi similar ao observado no desempenho para tempo de execução dentro dos quatro ambientes de execução.

5.4.3 Discussão dos Resultados

Tanto em tempo de execução como em consumo energético observou-se melhora de performance pelo uso de *offloading* computacional através do CoSMOS. Embora para resolução de 0.3 MP a performance entre os 4 ambientes demonstrem ser semelhantes, para valores superiores houve melhora significativa para as três formas de suporte.

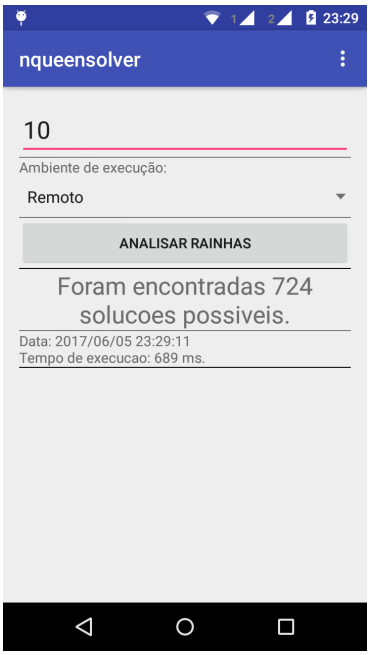
Entretanto, a instabilidade da infraestrutura de rede e o tráfego de dados entre dispositivo e servidor diminuiram a eficácia do uso desta técnica, podendo ser observado através da diferença de performance entre uso de servidor *cloud* via WiFi e via 3G. Embora acessem o mesmo servidor remoto, os meios diferentes de tráfego de dados impactaram na diferença de performance entre ambos.

O impacto causado por estes fatores foi minimizado pelo uso de suporte por servidor *cloudlet*, por este se localizar dentro da mesma rede de comunicação que o dispositivo. Isto pode ser observado através da melhora de performance tanto em tempo de execução como em consumo energético em relação a todos os demais.

5.5 Avaliação da Aplicação das N Rainhas

Foi desenvolvida uma aplicação móvel que busca encontrar o número de variação das soluções possíveis para o problema de N rainhas, onde o usuário define o número de rainhas ao qual deseja aplicar a busca. Sua tela inicial encontra-se ilustrada pela Figura 5.11.

Figura 5.11: Tela inicial da aplicação das N Rainhas



Elaborado pelo autor.

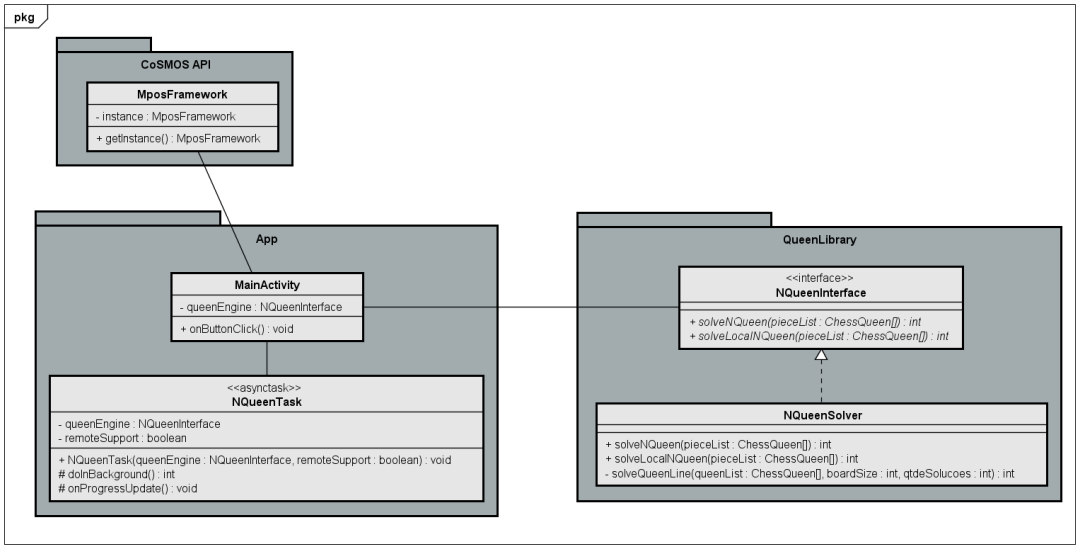
O problema de N rainhas é a extensão do problema das oito rainhas. Originalmente o problema propunha posicionar oito rainhas em um tabuleiro 8x8 sem que nenhuma rainha possa atacar diretamente outra rainha seja horizontal, vertical ou diagonalmente (BEZZEL, 1848).

Sua versão extendida busca posicionar um número arbitrário N de rainhas em um tabuleiro com dimensões igual ao número de rainhas, de NxN, mantendo a restrição de não permitir que as rainhas possam atacar umas às outras horizontal, vertical ou diagonalmente (GENT; JEFFERSON; NIGHTINGALE, 2017).

5.5.1 Estrutura da Aplicação Móvel

O diagrama de classes apresentado pela Figura 5.12 ilustra a arquitetura utilizada por esta aplicação, baseando-se no algoritmo básico de busca em largura para as soluções de posicionamento de um dado número N de rainhas em um tabuleiro de dimensões N por N.

Figura 5.12: Diagrama de classes da aplicação das N Rainhas



Elaborado pelo autor.

A aplicação das N Rainhas possui somente uma única tela de interface com o usuário, através da qual será informado o número de rainhas a ser analisada e selecionará o ambiente de execução da aplicação, se será realizada exclusivamente local ou se utilizará apoio por meio de plataforma remota.

Na parte inferior da tela serão informados o número de disposições distintas possíveis para o número de rainhas informado em um tabuleiro quadrado de dimensões iguais ao número informado, além de data de execução e tempo utilizado para o processamento.

5.5.2 Cenários de Execução

As execuçãos da aplicação das N rainhas foram divididas em oito séries de execução, cada qual constituído por dez ciclos, com cada ciclo contendo todos os números de rainhas entre 4 e 13. Todas as séries foram distribuídas igualmente pelos quatro ambientes de estudo: execução exclusivamente dentro do dispositivo móvel, apoio por servidor *Cloudlet* com conexão via WiFi, apoio por servidor *Cloud* com conexão via WiFi, e apoio por *Cloud* com conexão via 3G.

Para validar o algoritmo utilizado para resolução do problema das N Rainhas todos os resultados foram conferidos contra diversos sites relacionados à resolução do problema, e todas as execuções atingiram o valor esperado.

5.5.3 Resultados Obtidos

A seguir serão apresentados os resultados obtidos e análise da série de execuções para cada número de rainha sob três perspectivas: tempo médio de execução, consumo energético médio e taxa de uso das fases de validação em tomada de decisão.

Tempo de Execução

Com base nas informações de tempo de execução para cada chamada, duas perspectivas de análise de resultados foram organizadas: comparação de tempo médio para cada valor de número de rainhas e comparação com tempo médio geral em cada ambiente de execução.

A Tabela 5.10 apresenta informações sobre o tempo médio e desvio padrão de execução da aplicação das N rainhas para cada um dos números de rainhas entrados, sendo executado sob os quatro ambientes distintos. O tempo de execução de cada avaliação foi obtido através de mensuração pela própria aplicação.

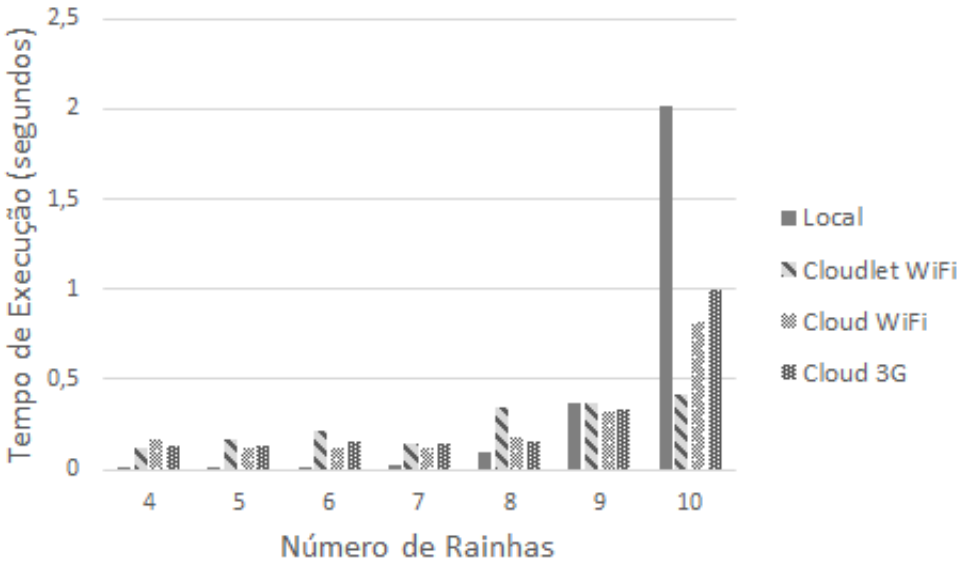
Tabela 5.10: Tempo médio em segundos de execução da aplicação das N rainhas para número de rainhas entre 4 e 13.

#	Suporte			
	Local	***Cloudlet* (WiFi)**	***Cloud* (WiFi)**	***Cloud* (3G)**
4	0,016 ± 0,052	0,117 ± 0,028	0,166 ± 0,198	0,126 ± 0,022
5	0,018 ± 0,030	0,171 ± 0,100	0,114 ± 0,024	0,129 ± 0,023
6	0,016 ± 0,031	0,216 ± 0,126	0,118 ± 0,033	0,150 ± 0,025
7	0,023 ± 0,026	0,146 ± 0,017	0,124 ± 0,053	0,145 ± 0,020
8	0,096 ± 0,037	0,345 ± 0,162	0,173 ± 0,022	0,160 ± 0,034
9	0,366 ± 0,018	0,368 ± 0,038	0,317 ± 0,020	0,335 ± 0,018
10	2,017 ± 0,043	0,410 ± 0,028	0,819 ± 0,144	0,998 ± 0,027
11	11,566 ± 0,352	0,494 ± 0,057	0,828 ± 0,079	3,717 ± 1,417
12	75,135 ± 0,445	0,968 ± 0,097	1,327 ± 0,233	5,089 ± 8,006
13	520,184 ± 2,566	3,274 ± 0,075	3,642 ± 0,658	10,716 ± 9,498

Elaborado pelo autor.

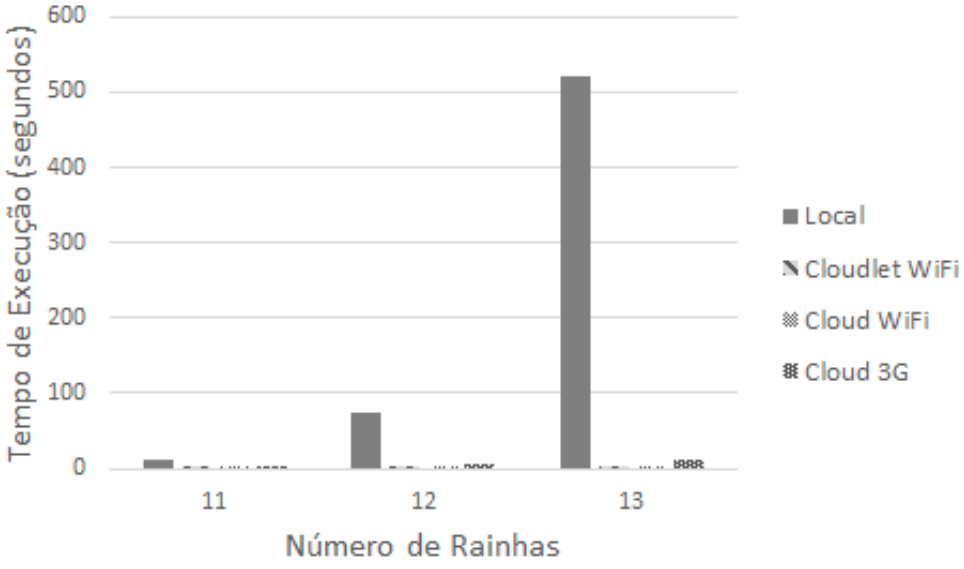
Os resultados da Tabela 5.10 relativos ao tempo de execução para cada número de rainhas encontram-se ilustrados pelos gráficos nas Figuras 5.13 e 5.14.

Figura 5.13: Gráfico de tempo de execução para a aplicação das N rainhas, com número de rainhas entre 4 e 10.



Elaborado pelo autor.

Figura 5.14: Gráfico de tempo de execução para a aplicação das N rainhas, com número de rainhas entre 11 e 13.



Elaborado pelo autor.

A Tabela 5.11 apresenta informações sobre tempo médio geral e desvio padrão de execução da aplicação das N rainhas para cada um dos quatro ambientes de execução.

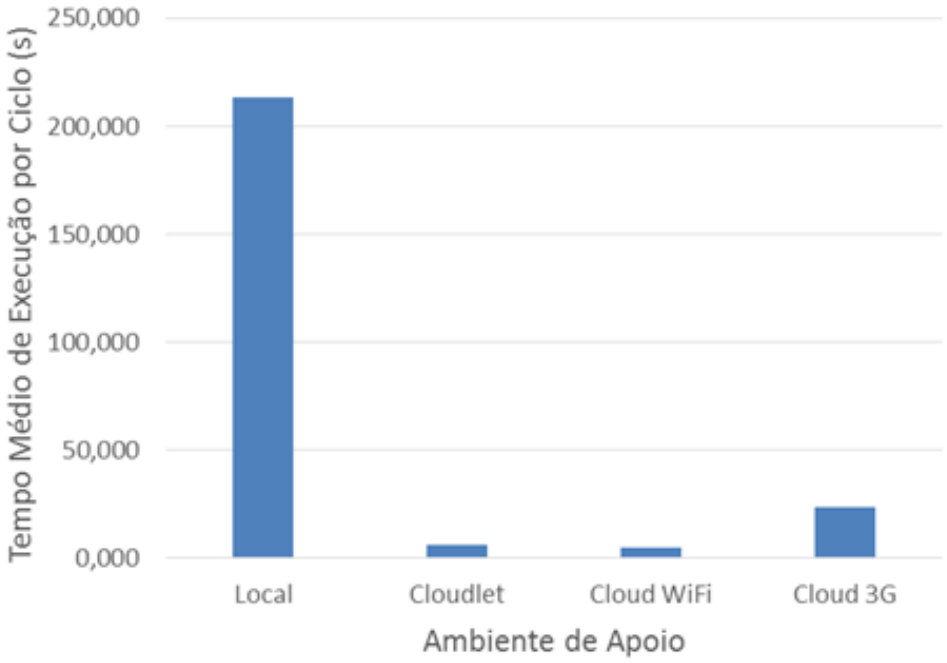
Tabela 5.11: Comparação de tempo médio de execução em segundos da aplicação das N rainhas por ciclo de avaliação em cada ambiente

Suporte	Tempo de Execução Total Médio por Ciclo (segundos)
Local no Dispositivo	213,091 ± 53,729
Plataforma *Cloudlet* (WiFi)	6,083 ± 0,914
Plataforma *Cloud* (WiFi)	5,049 ± 1,067
Plataforma *Cloud* (3G)	23,577 ± 5,024

Elaborado pelo autor.

Os resultados da Tabela 5.11 relativos ao tempo de execução médio geral encontra-se ilustrado pelo gráfico na Figura 5.15.

Figura 5.15: Gráfico de tempo de execução para a aplicação das N rainhas por ciclo de avaliação



Elaborado pelo autor.

A partir dos resultados obtidos nota-se que há tanto diferenças como similaridades de desempenho entre os quatro ambientes de execução. Enquanto a execução local oferece melhor performance de tempo que o uso de suporte para número de rainhas entre 4 e 9, para valores entre 10 e 13 esta demonstrou pior performance comparado aos demais.

Esta diferença se deve ao uso de *offloading* computacional, que melhorou performance de tempo em relação à execução local para o número de rainhas entre 10 e 13.

A diferença de performance observado para número de rainhas entre 4 e 9 se deve ao fato que o próprio uso do modelo também consumir recurso computacional, assim aumentando o tempo de execução.

Entretanto, a instabilidade da rede entre dispositivo móvel e servidor remoto impactou na estabilidade do tempo de execução. Este impacto pode ser observado através dos valores de desvio padrão para servidor *cloud*, especialmente para os números de rainha acima de 10.

Consumo Energético

A Tabela 5.12 apresenta informações sobre o consumo energético médio e desvio padrão de execução da aplicação das N rainhas para cada um dos números de rainhas entrados, sendo executado sob os quatro ambientes distintos. O consumo energético foi mensurado através da aplicação Trepn.

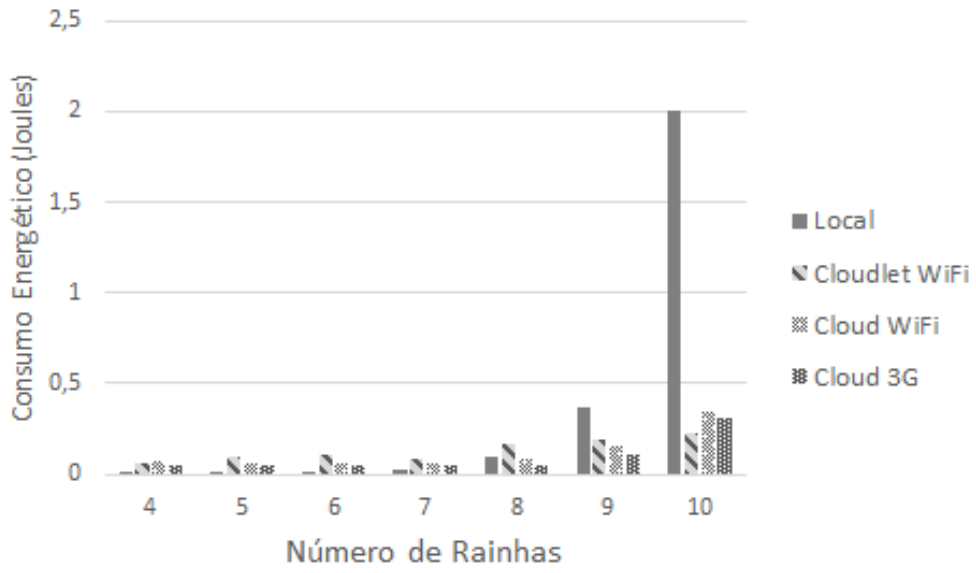
Tabela 5.12: Consumo energético médio em joules pela execução da aplicação das N rainhas

#	Suporte			
	Local	***Cloudlet* (WiFi)**	***Cloud* (WiFi)**	***Cloud* (3G)**
4	0,016 ± 0,025	0,060 ± 0,014	0,071 ± 0,057	0,046 ± 0,009
5	0,018 ± 0,016	0,091 ± 0,048	0,054 ± 0,016	0,042 ± 0,006
6	0,016 ± 0,031	0,110 ± 0,058	0,058 ± 0,020	0,051 ± 0,010
7	0,024 ± 0,013	0,079 ± 0,018	0,062 ± 0,028	0,047 ± 0,007
8	0,097 ± 0,024	0,168 ± 0,083	0,084 ± 0,011	0,050 ± 0,013
9	0,365 ± 0,055	0,188 ± 0,019	0,155 ± 0,032	0,102 ± 0,012
10	1,999 ± 0,169	0,221 ± 0,036	0,344 ± 0,154	0,310 ± 0,027
11	11,206 ± 1,236	0,275 ± 0,042	0,359 ± 0,075	1,324 ± 0,690
12	67,980 ± 2,214	0,540 ± 0,103	0,535 ± 0,066	1,998 ± 3,759
13	440,733 ± 10,982	1,608 ± 0,134	1,260 ± 0,240	2,563 ± 1,886

Elaborado pelo autor.

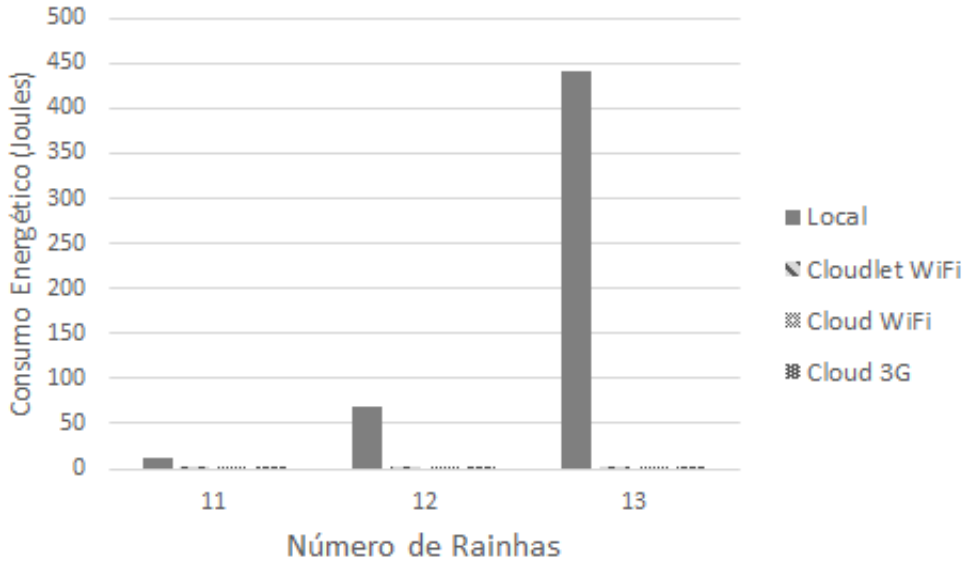
Os resultados apresentados pela Tabela 5.12 para consumo energético encontram-se ilustrados pelos gráficos nas Figuras 5.16 e 5.17.

Figura 5.16: Gráfico de consumo energético para a aplicação das N rainhas, com número de rainhas entre 4 e 10.



Elaborado pelo autor.

Figura 5.17: Gráfico de consumo energético para a aplicação das N rainhas, com número de rainhas entre 11 e 13.



Elaborado pelo autor.

A Tabela 5.13 apresenta informações sobre o consumo energético médio e desvio padrão geral da aplicação das N rainhas para cada um dos quatro ambientes de execução.

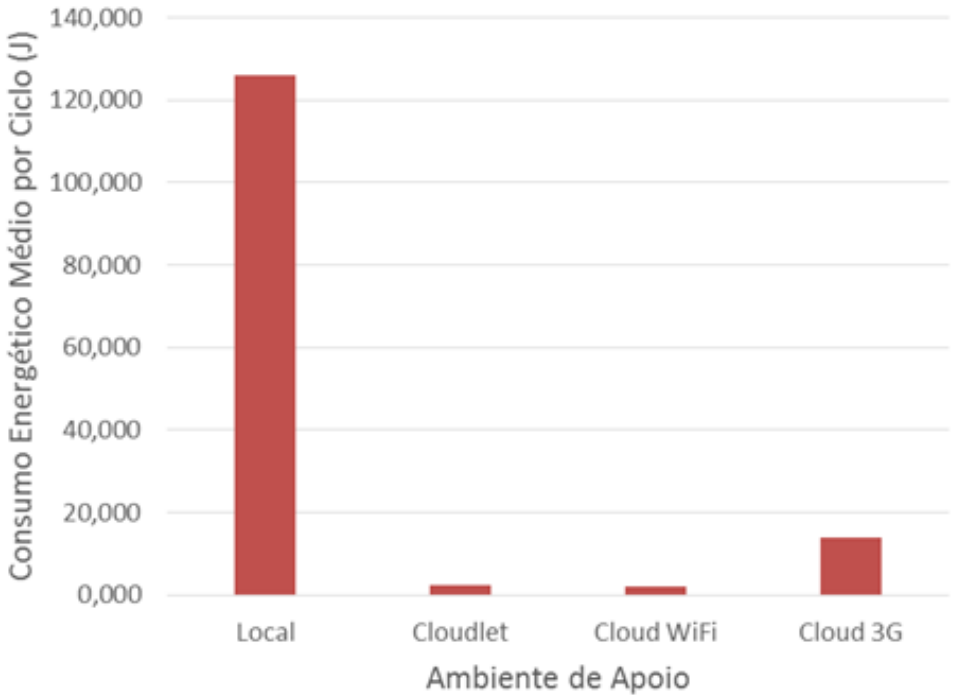
Tabela 5.13: Comparação de consumo energético médio em joules da aplicação das N rainhas por ciclo de avaliação em cada ambiente

Suporte	Consumo Energético Total Médio por Ciclo (Joule)
Local no Dispositivo	125,954 ± 34,802
Plataforma *Cloudlet* (WiFi)	2,300 ± 0,452
Plataforma *Cloud* (WiFi)	2,231 ± 0,371
Plataforma *Cloud* (3G)	13,972 ± 1,574

Elaborado pelo autor.

Os resultados da Tabela 5.13 relativos ao consumo energético médio geral encontra-se ilustrado pelo gráfico na Figura 5.18.

Figura 5.18: Gráfico de consumo energético para a aplicação das N rainhas por ciclo de avaliação



Elaborado pelo autor.

A partir dos resultados obtidos nota-se que, de maneira similar ao observado através da avaliação por tempo de execução, há tanto diferenças como similaridades de desempenho entre os quatro ambientes de execução. Enquanto a execução local oferece melhor performance em consumo energético que o uso de suporte para número de rainhas entre 4 e 8, para valores entre 9 e 13 esta demonstrou pior performance comparado aos demais.

Esta diferença se deve ao uso de *offloading* computacional, que melhorou performance de consumo em relação à execução local para o número de rainhas entre 9 e 13.

A diferença de performance observado para número de rainhas entre 4 e 8 se deve ao fato que o próprio uso do modelo também consome recurso computacional, assim aumentando o consumo energético.

Entretanto, a instabilidade da rede entre dispositivo móvel e servidor remoto impactou na estabilidade do consumo energético durante a execução, embora com menor impacto do que observado em tempo de execução. Este impacto no consumo pode ser observado através dos valores de desvio padrão para servidor *cloud*, especialmente para os números de rainha acima de 10.

Uso de *Offloading* Computacional

Também foi avaliada a taxa de uso de *offloading* computacional para servidor remoto, classificando entre execução local dentro do dispositivo e execução com suporte por plataforma remota.

A Tabela 5.14 apresenta informações sobre a taxa de *offloading* pelo CoSMOS para a aplicação das N rainhas, submetido a cada um dos números de rainhas entre 4 e 13, avaliando sob os três ambientes remotos distintos.

Tabela 5.14: Taxa de uso do *offloading* computacional para servidor remoto em cada ambiente para a aplicação das N rainhas.

#	**Suporte**					
	***Cloudlet* (WiFi)**		***Cloud* (WiFi)**		***Cloud* (3G)**	
	Local	**Remoto**	**Local**	**Remoto**	**Local**	**Remoto**
4	100%	0%	90%	10%	100%	0%
5	80%	20%	100%	0%	100%	0%
6	70%	30%	100%	0%	100%	0%
7	100%	0%	100%	0%	100%	0%
8	40%	60%	100%	0%	100%	0%
9	40%	60%	100%	0%	100%	0%
10	0%	100%	20%	80%	100%	0%
11	0%	100%	0%	100%	50%	50%
12	0%	100%	0%	100%	10%	90%
13	0%	100%	0%	100%	0%	100%

Elaborado pelo autor.

A partir destes dados observa-se que há nítida distinção de uso de *offloading* computacional de acordo com o número de rainhas, sendo empregado principalmente para valores acima de 10. Para valores abaixo de 8 o foco principal foi dado para execução local.

Entretanto há discrepância entre os ambientes para o uso de *offloading* com valores entre 8 e 10. Para servidor *cloudlet* os valores 8 e 9 foram divididos entre execução local e *offloading* para servidor remoto, enquanto para 10 priorizou-se execução remota.

Já para servidor *cloud* a diferença do uso de *offloading* mostrou-se mais nítida. Para tanto via WiFi como 3G o foco foi dado para execução local para 8 e 9, distinguindo-se para 10: enquanto via WiFi priorizou-se execução remota, para 3G priorizou-se execução local.

Estas diferenças de priorização de *offloading* devem-se à infraestrutura do meio de comunicação entre dispositivo e servidor remoto: enquanto servidor *cloudlet* possui maior estabilidade de conexão, servidor *cloud* sofre mais com instabilidade, especialmente para conexão via 3G.

Instabilidade da rede pode levar em aumento no tempo de tráfego de dados entre dispositivo e servidor remoto, que acarreta em degradação da execução da aplicação móvel ao aumentar o tempo de processamento.

Tais condições de rede implicam diretamente sobre o uso de *offloading* de acordo com os valores de número de rainhas: enquanto condições mais estáveis (servidor *cloudlet*) permitem *offloading* para valores mais baixos, condições menos estáveis (servidor *cloud*) favorecem seu uso somente para valores mais altos.

Tomada de Decisão

Também foi avaliado o uso das fases de validação na tomada de decisão de *offloading*, dividindo entre uso exclusivo da primeira fase e uso da primeira fase em conjunto com a segunda fase.

A Tabela 5.15 apresenta informações sobre a taxa de uso das fases de tomada de decisão pelo CoSMOS no suporte da aplicação das N rainhas, submetido a cada um dos números de rainhas de 4 a 13, sendo executado sob os três ambientes remotos distintos.

A Tabela 5.16 apresenta informações sobre taxa geral de uso das fases de validação de tomada de decisão de *offloading* no suporte da aplicação das N rainhas em cada um dos três ambientes remotos de execução.

Tabela 5.15: Taxa de uso das fases de tomada de decisão para cada número de rainhas e ambiente de execução para a aplicação das N rainhas.

#	Ambiente	1a Fase de Validação	1a e 2a Fases de Validação
4	Cloudlet WiFi	0%	100%
	Cloud WiFi	80%	20%
	Cloud 3G	100%	0%
5	Cloudlet WiFi	20%	80%
	Cloud WiFi	100%	0%
	Cloud 3G	100%	0%
6	Cloudlet WiFi	20%	80%
	Cloud WiFi	100%	0%
	Cloud 3G	100%	0%
7	Cloudlet WiFi	30%	70%
	Cloud WiFi	100%	0%
	Cloud 3G	100%	0%
8	Cloudlet WiFi	20%	80%
	Cloud WiFi	100%	0%
	Cloud 3G	100%	0%
9	Cloudlet WiFi	20%	80%
	Cloud WiFi	100%	0%
	Cloud 3G	100%	0%
10	Cloudlet WiFi	100%	0%
	Cloud WiFi	20%	80%
	Cloud 3G	100%	0%
11	Cloudlet WiFi	100%	0%
	Cloud WiFi	100%	0%
	Cloud 3G	60%	40%
12	Cloudlet WiFi	100%	0%
	Cloud WiFi	100%	0%
	Cloud 3G	90%	10%
13	Cloudlet WiFi	100%	0%
	Cloud WiFi	100%	0%
	Cloud 3G	100%	0%

Elaborado pelo autor.

Tabela 5.16: Taxa de uso das fases de tomada de decisão para cada ambiente de execução para a aplicação das N rainhas.

Ambiente	1a Fase de Validação	1a e 2a Fases de Validação
Cloudlet WiFi	51%	49%
Cloud WiFi	90%	10%
Cloud 3G	95%	5%

Elaborado pelo autor.

A partir destes dados observa-se que, enquanto houve uso equilibrado entre a primeira fase exclusivamente e o de ambas as fases em conjunto para tomada de decisão em execuções com suporte por servidor *cloudlet*, para os demais casos a taxa de uso exclusivo da primeira fase foi superior. Isso se deve à diferença de valores estimados para execução local e remoto em cada chamada dos componentes da aplicação.

Para número de rainhas abaixo de 11, onde há maior envolvimento da segunda fase de validação na tomada de decisão, através das Tabelas 5.10 e 5.12 nota-se que os valores obtidos das execuções local e remota são significantemente próximos para ambos os parâmetros de otimização, necessitando de análise mais minunciosa para avaliar qual o melhor direcionamento no uso do *offloading* computacional.

Nos demais casos observou-se uma diferença mais significativa entre os parâmetros para execuções local e remota, sendo suficiente a avaliação por valores estimados pela primeira fase para efetuar tomada de decisão.

Com base nestes resultados há indícios que o modelo de estimação de parâmetros adotado pela primeira fase de validação consegue guiar eficientemente o uso do *offloading* computacional, também sendo capaz de distinguir os casos que a discrepância entre as execuções local e remota são pequenas o suficiente para necessitar análise mais minunciosa, esta realizada pela segunda fase de validação.

5.5.4 Comparação com MpOS

Para efeito de comparação foi realizado estudo com o *framework* original MpOS (COSTA et al., 2015), aplicando seu apoio sobre a aplicação das N rainhas.

Foram realizados ao todo doze séries de execução, constituído por dez ciclos, cada ciclo contendo todos os números de rainhas entre 4 e 13. Todas as séries foram distribuídas igualmente pelos três ambientes de estudo: apoio por servidor *Cloudlet* com conexão via WiFi, apoio por *Cloud* com conexão via WiFi, e apoio por *Cloud* com conexão via 3G.

A seguir serão apresentadas as comparações entre os *frameworks* de suportes a partir dos resultados obtidos da série de execuções para cada número de rainha sob três perspectivas: tempo médio de execução e consumo energético médio.

Tempo de Execução

Para comparação entre os dois *frameworks* foram analisadas os valores médios dos parâmetros de tempo de execução pela aplicação das N rainhas com apoio por MpOS e CoSMOS, avaliando para cada número de rainhas nos três ambientes de execução.

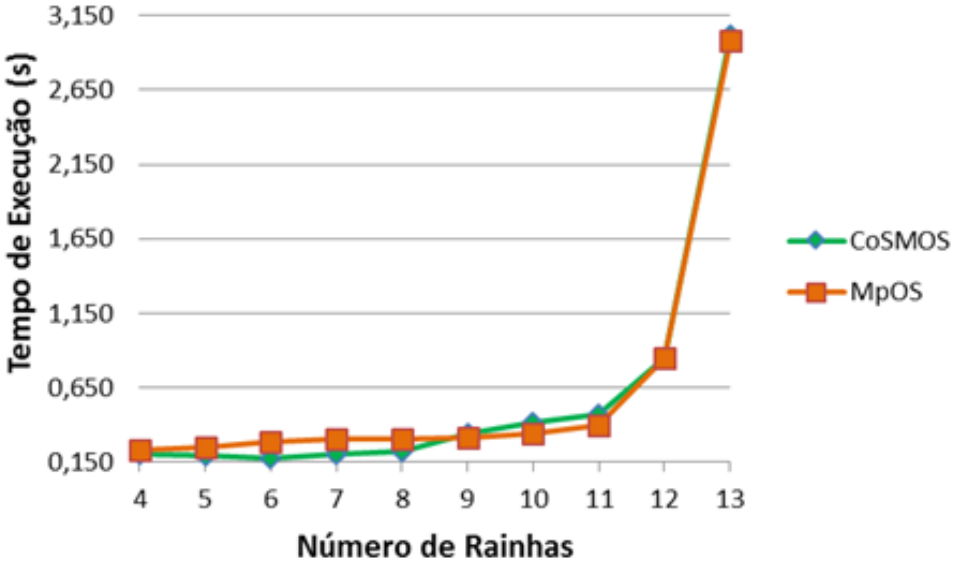
Os valores obtidos para tempo de execução encontram-se ilustrados pela Tabela 5.17 e pelos gráficos nas Figuras 5.19, 5.20 e 5.21.

Tabela 5.17: Tempo médio em segundos de execução da aplicação das N rainhas entre CoSMOS e MpOS

Nro de Rainhas	**Cloudlet (WiFi)**		**Cloud (WiFi)**		**Cloud (3G)**	
	MpOS	CoSMOS	MpOS	CoSMOS	MpOS	CoSMOS
4	0,230	0,200	0,675	0,122	0,471	0,119
5	0,244	0,190	0,632	0,141	1,094	0,117
6	0,281	0,175	0,650	0,134	0,984	0,103
7	0,308	0,202	0,661	0,150	0,292	0,102
8	0,300	0,222	0,654	0,170	1,586	0,144
9	0,313	0,340	0,638	0,317	0,512	0,307
10	0,340	0,412	0,708	0,341	1,519	1,038
11	0,393	0,472	1,241	0,480	3,255	2,669
12	0,853	0,849	3,981	0,706	7,745	7,162
13	2,981	3,021	3,864	2,489	76,20	11,814

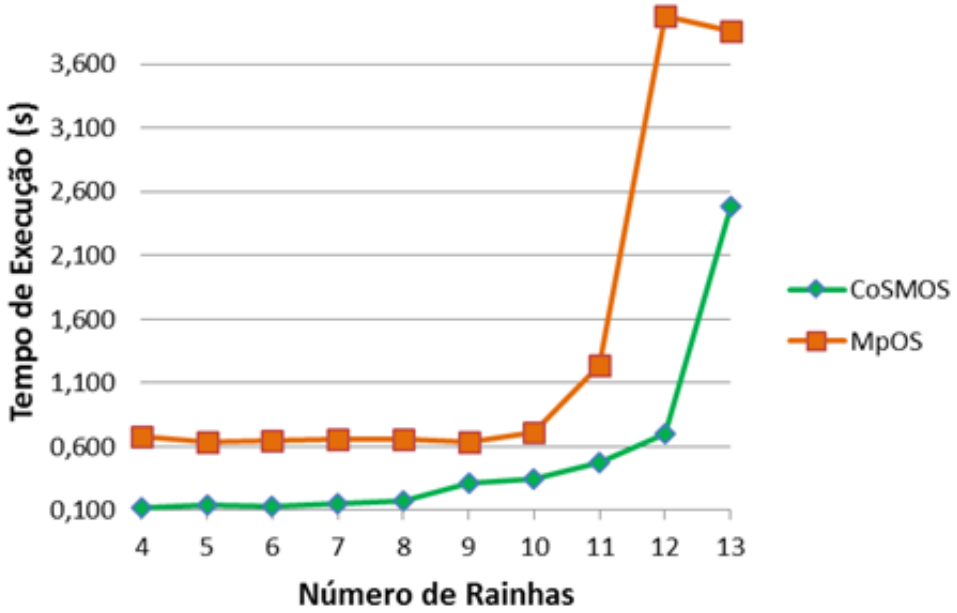
Elaborado pelo autor.

Figura 5.19: Gráfico de comparação de desempenho em tempo de execução com apoio por servidor *Cloudlet* via WiFi entre as plataformas MpOS e CoSMOS



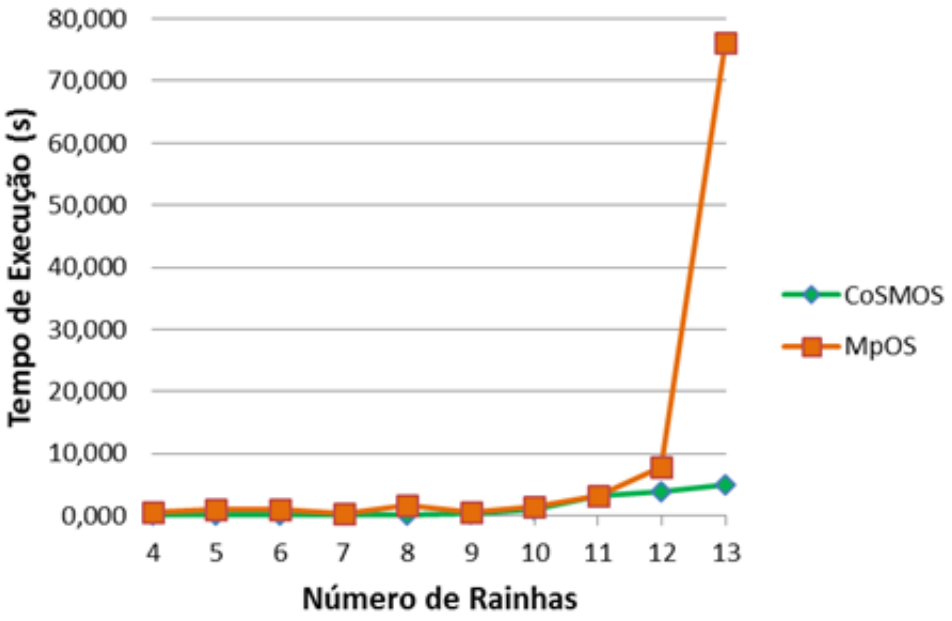
Elaborado pelo autor.

Figura 5.20: Gráfico de comparação de desempenho em tempo de execução com apoio por servidor *Cloud* via WiFi entre as plataformas MpOS e CoSMOS



Elaborado pelo autor.

Figura 5.21: Gráfico de comparação de desempenho em tempo de execução com apoio por servidor *Cloud* via 3G entre as plataformas MpOS e CoSMOS



Elaborado pelo autor.

Com base nos resultados obtidos, embora uso de CoSMOS e MpOS tenham obtido performances similares para suporte por servidor *cloudlet*, para suporte por servidor *cloud* a diferença entre ambos se tornou nítida, com CoSMOS manifestando diminuição consistente de tempo de execução em relação ao MpOS.

Isso se deve à diferença de política de uso do *offloading* computacional: enquanto MpOS busca utilizar *offloading* independentemente de oferecer melhora de performance em relação à execução local para condições ótimas de infraestrutura de rede, CoSMOS busca avaliar as diferenças de performance local e remota antes de optar pelo uso ou não da técnica. Essa diferença se torna mais notável no uso de suporte *cloud* via WiFi para número de rainhas entre 4 e 10, onde a execução local oferece vantagem sobre execução remota, tomando como base as Tabelas 5.17 e os valores para execução local da Tabela 5.10.

Para número de rainhas entre 11 e 13, apesar da execução remota oferecer vantagem sobre execução local, sofreu interferência da instabilidade do estado da infraestrutura, assim limitando severamente o uso do MpOS no uso de suporte *cloud*. Já CoSMOS, por se basear somente na diferença de performance ao invés de avaliar estado do ambiente, sofreu menor interferência da instabilidade.

Consumo Energético

Também foram analisadas os valores médios dos parâmetros de consumo energético pela aplicação das N rainhas com apoio por MpOS e CoSMOS, avaliando para cada número de rainhas nos três ambientes de execução.

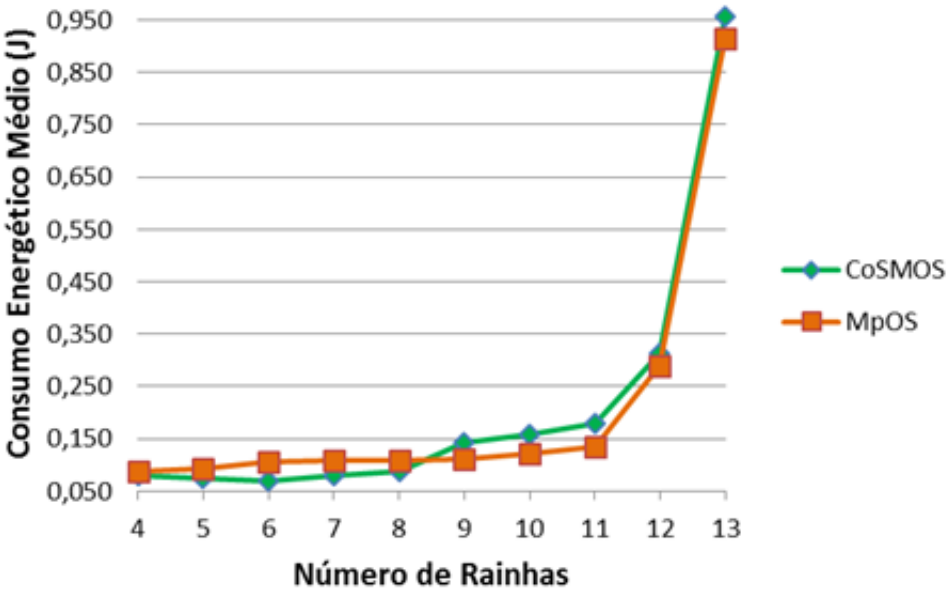
Os valores obtidos para consumo energético encontram-se ilustrados pela Tabela 5.18 e pelos gráficos nas Figuras 5.22, 5.23 e 5.24.

Tabela 5.18: Consumo energético em Joules pela execução da aplicação das N rainhas entre CoSMOS e MpOS

Nro de Rainhas	**Cloudlet (WiFi)**		**Cloud (WiFi)**		**Cloud (3G)**	
	MpOS	CoSMOS	MpOS	CoSMOS	MpOS	CoSMOS
4	0,087	0,078	0,255	0,060	0,153	0,063
5	0,093	0,075	0,230	0,072	0,295	0,064
6	0,105	0,068	0,250	0,068	0,268	0,062
7	0,109	0,079	0,237	0,076	0,082	0,059
8	0,108	0,088	0,246	0,084	0,443	0,081
9	0,112	0,143	0,231	0,163	0,128	0,178
10	0,121	0,159	0,257	0,168	0,399	0,595
11	0,136	0,180	0,548	0,232	1,037	1,588
12	0,288	0,314	2,066	0,326	3,385	4,159
13	0,914	0,958	1,189	0,981	43,599	7,123

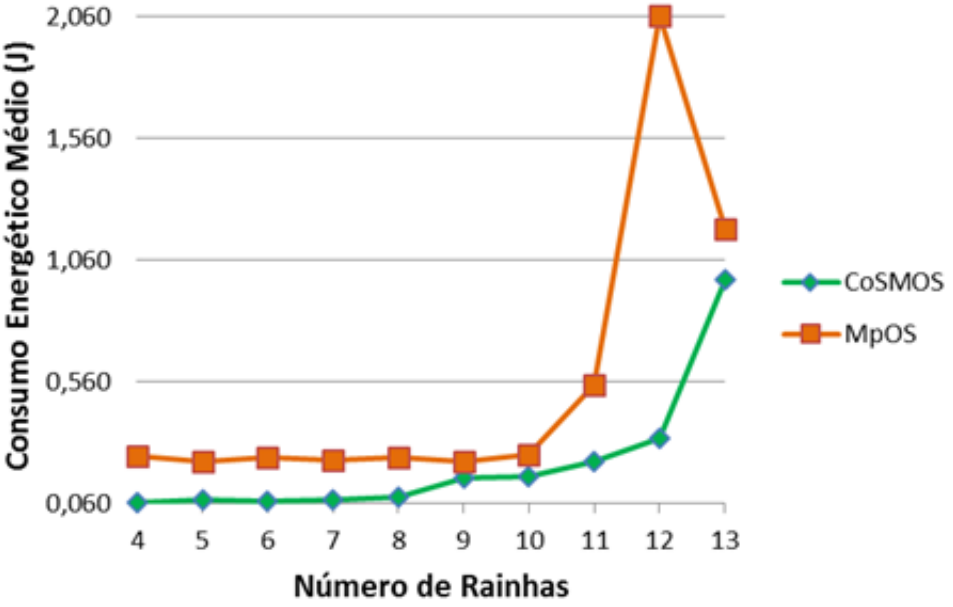
Elaborado pelo autor.

Figura 5.22: Gráfico de comparação de desempenho em consumo energético com apoio por servidor *Cloudlet* via WiFi entre as plataformas MpOS e CoSMOS



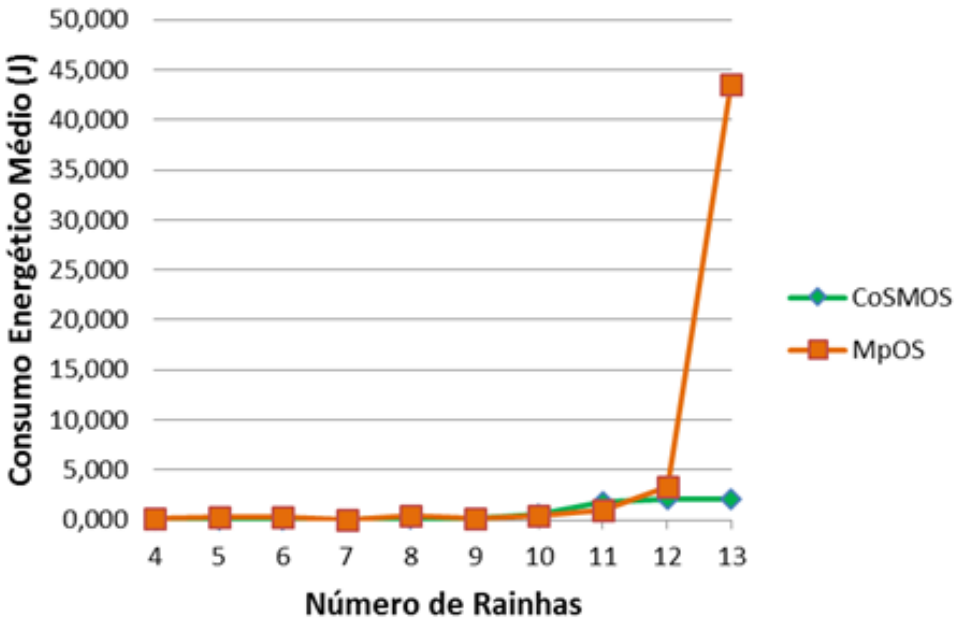
Elaborado pelo autor.

Figura 5.23: Gráfico de comparação de desempenho em consumo energético com apoio por servidor *Cloud* via WiFi entre as plataformas MpOS e CoSMOS



Elaborado pelo autor.

Figura 5.24: Gráfico de comparação de desempenho em consumo energético com apoio por servidor *Cloud* via 3G entre as plataformas MpOS e CoSMOS



Elaborado pelo autor.

Similarmente ao observado na comparação entre MpOS e CoSMOS em tempo de exeução, com base nos resultados obtidos para consumo energético, embora uso de CoSMOS e MpOS tenham obtido performances similares para suporte por servidor *cloudlet*, para suporte por servidor *cloud* a diferença entre ambos se tornou nítida, com CoSMOS manifestando diminuição consistente de tempo de execução em relação ao MpOS.

Isso também deve à diferença de política de uso do *offloading* computacional. Esta se torna mais notável no uso de suporte *cloud* via WiFi para número de rainhas entre 4 e 10, onde a execução local oferece vantagem sobre execução remota, tomando como base as Tabelas 5.18 e os valores para execução local da Tabela 5.12.

Para número de rainhas entre 11 e 13, apesar da execução remota oferecer vantagem sobre execução local, sofreu interferência da instabilidade do estado da infraestrutura, assim limitando severamente o uso do MpOS no uso de suporte *cloud*. Já CoSMOS, por se basear somente na diferença de performance ao invés de avaliar estado do ambiente, sofreu menor interferência da instabilidade.

5.5.5 Discussão dos Resultados

Dentro da comparação entre os quatro ambientes de execução somente para CoSMOS, nota-se melhora de desempenho através do uso de *offloading* computacional, especialmente para número de rainhas superior a 9.

Entretanto, embora o impacto do tráfego de dados e instabilidade da infraestrutura de comunicação entre dispositivo móvel e servidor remoto interfiram na eficácia do uso do CoSMOS, este se mostrou menor do que o impacto observado durante uso da aplicação *BenchImage* devido ao uso de dados significantemente menores em seu processamento do que da outra aplicação, implicando em menor tráfego de dados.

Através da comparação de CoSMOS e MpOS, este demonstrou melhor capacidade de adaptação da execução da aplicação do que o segundo, oferecendo melhoria de performance tanto em tempo de execução como em consumo energético. A diferença de políticas de avaliação de uso do *offloading* computacional entre os dois sistemas se tornou notável em ambos os parâmetros de otimização.

5.6 Resultados Gerais e Discussões

Embora o uso de *offloading* ofereça melhora de desempenho tanto em tempo de execução como em consumo energético para ambas as aplicações em relação à performance local, a otimização de seu uso através do modelo de tomada de decisão do CoSMOS melhorou sua eficácia, sendo capaz de otimizar a execução das aplicações mesmo quando submetido a ambientes com infraestrutura de comunicação instável e diferenças no tráfego de dados.

Entretanto, a instabilidade ainda afeta a capacidade de avaliação do modelo, potencialmente interferindo na estimação de parâmetros de otimização. Isso se deve à política de avaliação somente por tempo de execução e consumo energético, sem avaliar estado de conexão.

5.7 Considerações Finais do Capítulo

Este capítulo apresentou os resultados obtidos dos estudos de caso do modelo CoSMOS aplicado a duas aplicações móveis: *BenchImage* e uma implementação do problema das N rainhas. Ambas foram submetidas a execução em quatro ambientes distintos, analisando os resultados nos âmbitos de tempo de execução e consumo energético, avaliando tanto ganho de performance como capacidade de adaptação para otimização de desempenho.

Também foi realizada avaliação da acurácia de estimação de parâmetros de otimização por parte da primeira fase de validação com duas funções matemáticas, comparadas com resultados obtidos das execuções das aplicações das N rainhas e *BenchImage* em tempo de execução e consumo energético.

Por fim foi realizada comparação de performance entre CoSMOS e MpOS a execução da aplicação das N rainhas sob ótica de tempo de execução e consumo energético com suporte remoto.

Capítulo 6

CONCLUSÕES

Este capítulo tem como objetivo apresentar as principais contribuições que este trabalho oferece no campo de estudo de computação móvel nas nuvens, principais limitações enfrentadas durante a elaboração do trabalho, e sugestões de futuros trabalhos a serem realizados.

6.1 Contribuições do Trabalho

Esta dissertação apresentou um modelo para desenvolvimento de sistema de apoio à tomada de decisão multi-objetiva de *offloading* em plataforma de apoio à execução de aplicação móvel.

Apesar de CoSMOS ter obtido desempenho similar ao de *framework* mono-objetivo em suporte por servidor *cloudlet* via WiFi, seu desempenho mostrou-se melhor e mais consistente para suporte por servidor *cloud* tanto via WiFi como 3G. Com base nestes resultados pode-se afirmar que há indícios que tal modelo de tomada de decisão apresenta melhor capacidade de adaptação e otimização de execução de aplicações móveis para as constantes mudanças que ocorrem tanto no dispositivo móvel como na infraestrutura de rede de conexão entre dispositivo e servidor remoto, comparando a modelos de tomada de decisão mono-objetiva.

O trabalho contribuiu com aplicação de padrões de sistemas auto-adaptativos em desenvolvimento para a área de computação móvel nas nuvens, utilizando-os para lidar com a tomada de decisão multi-objetiva de *offloading* de funcionalidades através do modelo CoSMOS. O uso destes padrões ajudou na organização do processo de desenvolvimento deste trabalho, desde o levantamento de requisitos até sua implementação.

O CoSMOS, em conjunto com o *framework* MpOS, também demonstrou o impacto na melhora de performance de execução de aplicações móveis nos parâmetros de tempo de execução e consumo energético através de uso em casos práticos em ambientes distintos de execução.

6.2 Dificuldades Durante a Realização do Trabalho

Durante a elaboração deste trabalho, diversas limitações foram encontradas, com algumas delas impactando severamente durante sua realização.

Uma das limitações encontradas é a mensuração das dimensões dos parâmetros de entrada de cada chamada de funcionalidade. Devido ao fato da SDK do Android não dispor de ferramentas para mensurar dimensões de objetos, foi necessário transformar cada objeto em *ByteStreamArray*, como relatado na seção 4.1.2. Seu uso na mensuração das dimensões dos parâmetros impacta na performance geral do modelo ao aumentar *overhead* de execução durante o uso, aumentando tempo de execução e consumo energético. Entretanto não foi encontrado outra forma nativa da API Android para realizar tal tarefa.

Outra limitação encontrada é sobre versão de API do Android a ser utilizada neste trabalho. Apesar de informações sobre tensão da bateria do dispositivo estar disponível desde a API 5, informações sobre corrente elétrica da bateria do dispositivo foi disponibilizada somente a partir da API 21, o que implica no uso do modelo CoSMOS somente para aplicações móveis desenvolvidas para Android a partir da versão 5.1.

Outra limitação, de grande impacto, que foi encontrada durante o desenvolvimento deste trabalho relaciona-se a aplicação deste em casos de uso. A execução remota baseada somente em código-fonte Java *stateless* limitou o uso em aplicações móveis durante os estudos, ao dificultar o uso de seu suporte para aplicações que somente baseiam-se em oferecer interface para aplicações que não sejam nativas em Android, tais como Pachi (BOUZY; CAZENAVE, 2001) e GnuGo para Go, e StockFish para xadrez.

Estes casos baseiam-se na execução de aplicações *engine* que guardam o estado atual para realizar processamento do estado seguinte, indo contra a natureza do formato sem estado oferecido pela plataforma.

6.3 Trabalhos Futuros

Este trabalho poderá servir como base para novos estudos, tais como:

- Utilizar meios alternativos de demarcar se componentes já foram executadas local e/ou remotamente com determinadas configurações;
- Realizar estudos de caso com conexão 4G;

- Realizar estudos com aplicação do modelo CoSMOS para outras formas de acoplamento em aplicações móveis;
- Incorporar uso de novos parâmetros de otimização para tomada de decisão de *offloading*, tais como latência de rede e tráfego de dados entre dispositivo móvel e servidor na nuvem;
- Estudar outras abordagens para estimação dos parâmetros de otimização;
- Estudar abordagens para representação dos parâmetros de otimização dentro da etapa de tomada de decisão que permita fácil escalabilidade, ou seja, que comportem adições de novos parâmetros sem comprometer sua estrutura básica. Uma posibilidade seria através de algoritmos evolucionários (DEB, 2011).

REFERÊNCIAS

AHMED, E.; GANI, A.; SOOKHAK, M.; HAMID, S. H. A.; XIA, F. Application optimization in mobile cloud computing: Motivation, taxonomies, and open challenges. *Journal of Network and Computer Applications*, v. 52, p. 52 – 68, 2015.

AHMED, E.; NAVEED, A.; HAMID, S. H. A.; GANI, A.; SALAH, K. Formal analysis of seamless application execution in mobile cloud computing. *The Journal of Supercomputing*, p. 1–27, 2017. ISSN 1573-0484. Disponível em: ⟨http://dx.doi.org/10.1007/s11227-017-2028-4⟩.

BAHTOVSKI, A.; GUSEV, M. Cloudlet challenges. *Procedia Engineering*, v. 69, p. 704 – 711, 2014. ISSN 1877-7058. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S1877705814002914⟩.

BEZZEL, F. Proposal of eight queens problem. *Berliner Schachzeitung*, v. 3, p. 363, 1848.

BOUZY, B.; CAZENAVE, T. Computer go: An ai oriented survey. *Artificial Intelligence*, v. 132, n. 1, p. 39 – 103, 2001. ISSN 0004-3702. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0004370201001278⟩.

CHEN, T.; FANIYI, F.; BAHSOON, R.; LEWIS, P. R.; YAO, X.; MINKU, L. L.; ESTERLE, L. The handbook of engineering self-aware and self-expressive systems. *Computing Research Repository (CoRR)*, abs/1409.1793, 2014. Acesso em: 04 de agosto de 2016. Disponível em: ⟨http://arxiv.org/abs/1409.1793⟩.

CHEN, X.; CHEN, S.; ZENG, X.; ZHENG, X.; ZHANG, Y.; RONG, C. Framework for context-aware computation offloading in mobile cloud computing. *Journal of Cloud Computing*, v. 6, n. 1, p. 1, 2017. ISSN 2192-113X. Disponível em: ⟨http://dx.doi.org/10.1186/s13677-016-0071-y⟩.

CHUN, B.-G.; IHM, S.; MANIATIS, P.; NAIK, M.; PATTI, A. Clonecloud: Elastic execution between mobile device and cloud. In: *Proceedings of the Sixth Conference on Computer Systems*. [S.l.: s.n.], 2011. p. 301–314.

COMSCORE. *The U.S. Mobile App Report*. 2014. Acesso em: 18 de maio de 2016. Disponível em: ⟨https://www.comscore.com/Insights/Presentations-and-Whitepapers/2014/The-US-Mobile-App-Report⟩.

COSTA, P. B.; REGO, P. A. L.; ROCHA, L. S.; TRINTA, F. A. M.; SOUZA, J. N. de. Mpos: A multiplatform offloading system. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. [S.l.: s.n.], 2015. (SAC '15), p. 577–584.

CUERVO, E.; BALASUBRAMANIAN, A.; CHO, D.-k.; WOLMAN, A.; SAROIU, S.; CHANDRA, R.; BAHL, P. Maui: Making smartphones last longer with code offload. In: *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. [S.l.: s.n.], 2010. (MobiSys '10), p. 49–62.

DEB, K. Multi-objective optimisation using evolutionary algorithms: An introduction. In: ____. *Multi-objective Evolutionary Optimisation for Product Design and Manufacturing*. London: Springer London, 2011. p. 3–34. ISBN 978-0-85729-652-8. Disponível em: ⟨https://doi.org/10.1007/978-0-85729-652-8_1⟩.

DUTT, N.; JANTSCH, A.; SARMA, S. Toward smart embedded systems: A self-aware system-on-chip (soc) perspective. *ACM Trans. Embed. Comput. Syst.*, ACM, New York, NY, USA, v. 15, n. 2, p. 22:1–22:27, fev. 2016. ISSN 1539-9087. Disponível em: ⟨http://doi.acm.org.ez31.periodicos.capes.gov.br/10.1145/2872936⟩.

DUVAL, S.; WICKLUND, R. A. *A theory of objective self-awareness*. [S.l.: s.n.], 1972. 238 p.

FERNANDO, N.; LOKE, S. W.; RAHAYU, W. Mobile cloud computing: A survey. *Future Generation Computer Systems*, v. 29, n. 1, p. 84 – 106, 2013.

FILHO, D. F.; NUNES, F.; ROCHA, E. C. da; SANTOS, M. L.; BATISTA, M.; JúNIOR, J. A. S. O que fazer e o que não fazer com a regressão: pressupostos e aplicações do modelo linear de mínimos quadrados ordinários (mqo). *Revista Política Hoje*, v. 20, n. 1, 2011. ISSN 0104-7094.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-63361-2.

GENT, I. P.; JEFFERSON, C.; NIGHTINGALE, P. Complexity of n-queens completion. *J. Artif. Intell. Res.*, v. 59, p. 815–848, 2017. Disponível em: ⟨https://doi.org/10.1613/jair.5512⟩.

HINCHEY, M. G.; STERRITT, R. Self-managing software. *Computer*, v. 39, n. 2, p. 107–109, Feb 2006. ISSN 0018-9162.

HOQUE, M. A.; SIEKKINEN, M.; KHAN, K. N.; XIAO, Y.; TARKOMA, S. Modeling, profiling, and debugging the energy consumption of mobile devices. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 48, n. 3, p. 39:1–39:40, dez. 2015. ISSN 0360-0300. Disponível em: ⟨http://doi.acm.org/10.1145/2840723⟩.

IBM (Ed.). *An Architectural Blueprint for Autonomic Computing*. [S.l.], 2005.

KEMP, R.; PALMER, N.; KIELMANN, T.; BAL, H. Cuckoo: A computation offloading framework for smartphones. In: ____. *Mobile Computing, Applications, and Services: Second International ICST Conference, MobiCASE 2010, Santa Clara, CA, USA, October 25-28, 2010, Revised Selected Papers*. [S.l.]: Springer Berlin Heidelberg, 2012. p. 59–79.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. *Computer*, v. 36, n. 1, p. 41–50, Jan 2003.

KHAN, A. u. R.; OTHMAN, M.; MADANI, S. A.; KHAN, S. U. A survey of mobile cloud computing application models. *IEEE Communications Surveys Tutorials*, v. 16, n. 1, p. 393–413, First 2014. ISSN 1553-877X.

KHAN, A. u. R.; OTHMAN, M.; XIA, F.; KHAN, A. N. Context-aware mobile cloud computing and its challenges. *IEEE Cloud Computing*, v. 2, n. 3, p. 42–49, May 2015. ISSN 2325-6095.

KIM, H. *Smart Offloading Algorithms and Profiling Techniques for Cloud Computing*. 2012. Acesso em: 13 de setembro de 2016. Disponível em: ⟨http://www.star.cc.gatech.edu/prj_smart.php?menu=research⟩.

KOSTA, S.; AUCINAS, A.; HUI, P.; MORTIER, R.; ZHANG, X. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: *INFOCOM, 2012 Proceedings IEEE*. [S.l.: s.n.], 2012. p. 945–953. ISSN 0743-166X.

KOVACHEV, D.; YU, T.; KLAMMA, R. Adaptive computation offloading from mobile devices into the cloud. In: *Proceedings of the 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*. Washington, DC, USA: IEEE Computer Society, 2012. (ISPA '12), p. 784–791. ISBN 978-0-7695-4701-5. Disponível em: ⟨http://dx.doi.org/10.1109/ISPA.2012.115⟩.

KUMAR, K.; LIU, J.; LU, Y.-H.; BHARGAVA, B. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, Springer-Verlag New York, Inc., v. 18, n. 1, p. 129–140, fev. 2013. ISSN 1383-469X.

LEWIS, P. R.; CHANDRA, A.; FANIYI, F.; GLETTE, K.; CHEN, T.; BAHSOON, R.; TORRESEN, J.; YAO, X. Architectural aspects of self-aware and self-expressive computing systems: From psychology to engineering. *Computer*, v. 48, n. 8, p. 62–70, Aug 2015.

LEWIS, P. R.; CHANDRA, A.; PARSONS, S.; ROBINSON, E.; GLETTE, K.; BAHSOON, R.; TORRESEN, J.; YAO, X. A survey of self-awareness and its application in computing systems. In: *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2011 Fifth IEEE Conference on*. [S.l.: s.n.], 2011. p. 102–107.

NAQVI, N. Z.; DEVLIEGHERE, J.; PREUVENEERS, D.; BERBERS, Y. Mascot: Self-adaptive opportunistic offloading for cloud-enabled smart mobile applications with probabilistic graphical models at runtime. In: *2016 49th Hawaii International Conference on System Sciences (HICSS)*. [S.l.: s.n.], 2016. p. 5701–5710.

NARAYANAN, D.; FLINN, J.; SATYANARAYANAN, M. Using history to improve mobile application adaptation. In: *Proceedings Third IEEE Workshop on Mobile Computing Systems and Applications*. [S.l.: s.n.], 2000. p. 31–40.

NEISSER, U. The roots of self-knowledge: Perceiving self, it, and thoua. *Annals of the New York Academy of Sciences*, Blackwell Publishing Ltd, v. 818, n. 1, p. 19–33, 1997.

PAMBORIS, A. *Mobile Code Offloading for Multiple Resources*. Tese (Doutorado) — Imperial College London, mar 2014.

PARANJOTHI, A.; KHAN, M. S.; NIJIM, M. Survey on three components of mobile cloud computing: Offloading, distribution and privacy. v. 5, p. 1–31, 04 2017.

PILLAI, P. S.; MUMMERT, L. B.; SCHLOSSER, S. W.; SUKTHANKAR, R.; HELFRICH, C. J. Slipstream: Scalable low-latency interactive perception on streaming data. In: *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital*

Audio and Video. New York, NY, USA: ACM, 2009. (NOSSDAV '09), p. 43–48. ISBN 978-1-60558-433-1. Disponível em: ⟨http://doi.acm.org/10.1145/1542245.1542256⟩.

RA, M.-R.; SHETH, A.; MUMMERT, L.; PILLAI, P.; WETHERALL, D.; GOVINDAN, R. Odessa: Enabling interactive perception applications on mobile devices. In: *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*. New York, NY, USA: ACM, 2011. (MobiSys '11), p. 43–56. ISBN 978-1-4503-0643-0. Disponível em: ⟨http://doi.acm.org/10.1145/1999995.2000000⟩.

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, ACM, New York, NY, USA, v. 4, n. 2, p. 14:1–14:42, maio 2009. ISSN 1556-4665. Disponível em: ⟨http://doi.acm.org/10.1145/1516533.1516538⟩.

SANAEI, Z.; ABOLFAZLI, S.; GANI, A.; KHOKHAR, R. H. Tripod of requirements in horizontal heterogeneous mobile cloud computing. *CoRR*, abs/1205.3247, 2012.

VERBELEN, T.; SIMOENS, P.; TURCK, F. D.; DHOEDT, B. Aiolos: Middleware for improving mobile application performance through cyber foraging. *Journal of Systems and Software*, v. 85, n. 11, p. 2629 – 2639, 2012. ISSN 0164-1212. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0164121212001641⟩.

WANG, Y.; CHEN, I.-R.; WANG, D.-C. A survey of mobile cloud computing applications: Perspectives and challenges. *Wirel. Pers. Commun.*, Kluwer Academic Publishers, v. 80, n. 4, p. 1607–1623, fev. 2015.

ZHOU, B.; BUYYA, R. Augmentation techniques for mobile cloud computing: A taxonomy, survey, and future directions. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 51, n. 1, p. 13:1–13:38, jan. 2018. ISSN 0360-0300. Disponível em: ⟨http://doi-acm-org.ez31.periodicos.capes.gov.br/10.1145/3152397⟩.