UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

CAROLINA LUIZA CHAMAS

Consumo de energia em dispositivos móveis Android: análise das estratégias
de comunicação utilizadas em *Computation Offloading*

São Paulo

2018

CAROLINA LUIZA CHAMAS

**Consumo de energia em dispositivos móveis Android: análise das estratégias
de comunicação utilizadas em *Computation Offloading***

Dissertação apresentada à Escola de Artes,
Ciências e Humanidades da Universidade de
São Paulo para obtenção do título de Mestre
em Ciências pelo Programa de Pós-graduação
em Sistemas de Informação.

Área de concentração: Metodologia e
Técnicas da Computação

Versão corrigida contendo as alterações
solicitadas pela comissão julgadora em 14
de Dezembro de 2017. A versão original
encontra-se em acervo reservado na Biblio-
teca da EACH-USP e na Biblioteca Digital
de Teses e Dissertações da USP (BDTD), de
acordo com a Resolução CoPGr 6018, de 13
de outubro de 2011.

Orientador: Prof. Dr. Marcelo Medeiros Eler

Coorientador: Prof. Dr. Daniel Cordeiro

São Paulo

2018

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

CATALOGAÇÃO-NA-PUBLICAÇÃO

(Universidade de São Paulo. Escola de Artes, Ciências e Humanidades. Biblioteca)
CRB-8 4625

Chamas, Carolina Luiza

Consumo de energia em dispositivos móveis Android : análise das estratégias de comunicação utilizadas em *Computation Offloading* / Carolina Luiza Chamas ; orientador, Marcelo Medeiros Eler ; coorientador, Daniel Cordeiro. - São Paulo, 2018.
96 p. : il.

Dissertação (Mestrado em Ciências) - Programa de Pós-Graduação em Sistemas de Informação, Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, em 2017.

Versão corrigida

1. Computação móvel. 2. Economia de energia. 3. *Computation offloading*. I. Eler, Marcelo Medeiros, orient. II. Cordeiro Daniel, coorient. III. Título.

CDD 22.ed.— 004.145

Dissertação de autoria de Carolina Luiza Chamas, sob o título “**Consumo de energia em dispositivos móveis Android: análise das estratégias de comunicação utilizadas em *Computation Offloading***”, apresentada à Escola de Artes, Ciências e Humanidades da Universidade de São Paulo, para obtenção do título de Mestre em Ciências pelo Programa de Pós-graduação em Sistemas de Informação, na área de concentração Metodologia e Técnicas da Computação, aprovada em 14 de Dezembro de 2017 pela comissão julgadora constituída pelos doutores:

Prof. Dr. Marcelo Medeiros Eler

Universidade de São Paulo

Presidente

Profa. Dra. Fátima de Lima Procópio Duarte Figueiredo

Pontifícia Universidade Católica de Minas Gerais

Profa. Dra. Itana Stiubiener

Universidade Federal do ABC

Prof. Dr. Luciano Vieira de Araújo

Universidade de São Paulo

Dedico este trabalho às pessoas mais presentes em minha vida: À minha mãe pelo apoio, compreensão em períodos de ausência e por fazer o impossível por minha educação. Ao meu marido Leandro Colevati pelo apoio incondicional e constante incentivo. Ao meu orientador Prof Dr. Marcelo Eler e co-orientador Prof Dr. Daniel Cordeiro, pela confiança, paciência e excelente orientação.

Agradecimentos

Agradeço por ter pessoas tão especiais ao meu lado, sem as quais certamente eu não teria dado conta de finalizar esta dissertação!

À minha mãe que sempre se preocupou com minha educação, sempre fazendo de tudo para que eu tivesse bons estudos e que vibrou comigo desde minha aprovação!

Ao meu marido, Leandro, por ser tão importante na minha vida. Devido a seu companheirismo, amizade, paciência, compreensão, apoio, alegria e amor, este trabalho pôde ser concretizado!

Agradeço também aos meus sogros, Nelson e Nadir pelo apoio e carinho!

Aos meus amigos do mestrado, pelos momentos que divididos juntos, especialmente à Ligia que tornou mais leve essa fase, dividindo momentos de angústias e alegrias. Ao Lenin por dividir seu conhecimento na matéria de Algoritmos.

À professora Karina Valdivia pela paciência e disponibilidade em ajudar com a matéria de Algoritmos.

Aos professores Fátima e Luciano Araújo que foram referências para mim com a matéria de inovação, minha visão de mundo mudou depois de suas aulas!

À minha professora Sandra por entender, sem questionar, todas as aulas desmarcadas e remarcadas.

À Ericsson, em especial ao Celso Martinez e a Amarilis Nemer que me apoiaram e viabilizaram esta pós-graduação e também ao antigo time 14 e 15 por compreenderem alguns períodos de ausência.

Finalmente, gostaria de agradecer ao meu orientador Prof. Dr. Marcelo Eler e co-orientador Prof. Dr. Daniel Cordeiro por abrirem as portas para que eu pudesse realizar mais uma conquista, um mestrado na USP. Vocês me proporcionaram mais que a busca de conhecimento técnico e científico, mas uma lição de vida.

“Lembre-se de que ninguém vence sozinho. Tenha gratidão no coração e reconheça prontamente aqueles que o ajudaram.” (Johnny De’ Carli)

“Apesar dos nossos defeitos, precisamos enxergar que somos pérolas únicas no teatro da vida e entender que não existem pessoas de sucesso e pessoas fracassadas. O que existem são pessoas que lutam pelos seus sonhos ou desistem deles.”

(Augusto Cury)

Resumo

CHAMAS, Carolina Luiza. **Consumo de energia em dispositivos móveis Android: análise das estratégias de comunicação utilizadas em *Computation Offloading***. 2018. 96 f. Dissertação (Mestrado em Ciências) – Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, São Paulo, 2017.

Os dispositivos móveis passaram por grandes transformações na última década e tornaram-se complexos computadores dotados de grande poder de processamento e memória, além de prover aos usuários diversos recursos como sensores e câmeras de alta resolução. O uso de dispositivos móveis para diversas tarefas aumentou consideravelmente, o que levantou uma grande preocupação com o alto consumo de energia desses dispositivos. Portanto, estudos tem sido realizados no sentido de encontrar soluções para diminuir o custo de energia das aplicações que executam em dispositivos móveis. Uma das alternativas mais utilizadas é o *computation offloading*, cujo objetivo é transferir a execução de uma tarefa para uma plataforma externa com o intuito de aumentar desempenho e reduzir consumo de recursos, como a bateria, por exemplo. Decidir sobre usar ou não esta técnica implica entender a influência de fatores como a quantidade de dados processados, a quantidade de computação envolvida, e o perfil da rede. Muitos estudos tem sido realizados para estudar a influência de diversas opções de rede wireless, como 3G, 4G e Wifi, mas nenhum estudo investigou a influência das escolhas de comunicação no custo de energia. Portanto, o objetivo deste trabalho é apresentar uma investigação sobre a influência da quantidade de dados, da quantidade de computação e dos protocolos de comunicação ou estilo arquitetural no consumo de energia quando a técnica de *computation offloading* é utilizada. Neste estudo, foram comparados REST, SOAP, Socket e RPC na execução de algoritmos de ordenação de diferentes complexidades aplicados sobre vetores de diversos tamanhos e tipos de dados. Os resultados mostram que a execução local é mais econômica com algoritmos menos complexos, pequeno tamanho de entrada e tipo de dados menos complexos. Quando se trata de execução remota, o REST é a escolha mais econômica seguida por Socket. Em geral, REST é mais econômico com vetores do tipo Object, independentemente da complexidade do algoritmo e tamanho do vetor, enquanto Socket é mais econômico com entradas maiores e com vetores de tipos primitivos, como Int e Float.

Palavras-chaves: Computação móvel. Economia de energia. *Computation offloading*.

Abstract

CHAMAS, Carolina Luiza. **Energy consumption on Android mobile devices:** communication strategies analysis used in Computation Offloading. 2018. 96 p.
Dissertation (Master of Science) – School of Arts, Sciences and Humanities, University of São Paulo, São Paulo, 2017.

Mobile devices have significantly changed in the last decade and they become complex computer machines equipped with large processing power and memory. Moreover, they provide users with several resources such as sensors and high resolution cameras. The usage of mobile devices has significantly increased in the past years, which raised an important concern regarding the high energy consumption. Therefore, several investigations have been conducted aiming at finding solutions to reduce the energy cost of mobile applications. One of the most used strategy is called computation offloading, whose main goal is to transfer the execution of a task to an external platform aiming at increasing performance and reducing resource consumption, including the battery. Deciding towards offloading certain tasks requires to understand the influence of the amount of data, amount of computation, and the network profile. Several studies have investigated the influence of different wireless flavours, such as 3G, 4G and wifi, but no study has investigated the influence of the communication choices on the energy cost. Therefore, the purpose of this research project is to present an investigation on the influence of the amount of data, amount of computation and the communication protocols and architectural style on the energy consumption in the context of the computation offloading technique. In this study, we compare REST, SOAP, Socket and RPC when executing algorithms of different complexities and different input sizes and types. Results show that local execution is more economic with less complex algorithms and small input data. When it comes to remote execution, REST is the most economic choice followed by Socket. In general, REST is the most economic choice when applied on Object type arrays, regardless the complexity and size, while Socket is the most economic choice with large arrays and primitive types such as integers and floats.

Keywords: Mobile computing. Energy saving. Computation offloading.

Lista de figuras

Figura 1 – Análise de custo benefício e tomada de decisão para <i>computation offloading</i>	26
Figura 2 – Representação esquemática de aplicação distribuída baseada em Socket	30
Figura 3 – Representação de uma mensagem SOAP, composta pelos elementos: envelope, cabeçalho e corpo	32
Figura 4 – Representação da estrutura arquitetural do RMI composta por <i>stubs</i> e <i>skeletons</i> , referências remotas e transporte	37
Figura 5 – Diagrama de atividade do aplicativo automatizado utilizado nos experi- mentos	45
Figura 6 – Consumo de energia dos tipos de dados de entrada das execuções locais com o celular Galaxy S5	51
Figura 7 – Comparação do consumo de energia das execuções locais e as execuções remotas (<i>offloading</i>) mais econômica no Galaxy S5 com relação aos tipos dos dados	52
Figura 8 – Consumo de energia das execuções remotas (<i>offloading</i>) no Galaxy S5 com relação aos tipos dos dados	53
Figura 9 – Consumo de energia dos tamanhos de entrada das execuções locais com o celular Galaxy S5	54
Figura 10 – Comparação do consumo de energia das execuções locais e as execuções remotas (<i>offloading</i>) mais econômica no Galaxy S5 com relação aos tamanhos de entrada do algoritmo	54
Figura 11 – Consumo de energia das execuções remotas (<i>offloading</i>) no Galaxy S5 com relação aos tamanhos de entrada	55
Figura 12 – Consumo de energia das complexidades dos algoritmos das execuções locais com o celular Galaxy S5	56
Figura 13 – Comparação do consumo de energia das execuções locais e as execuções remotas (<i>offloading</i>) mais econômica no Galaxy S5 com relação as complexidades assintóticas	57
Figura 14 – Consumo de energia das execuções remotas (<i>offloading</i>) no Galaxy S5 com relação as complexidades assintóticas	57

Figura 15 – Gráficos da comparação do consumo de energia das execuções locais e as execuções remotas (<i>offloading</i>) do Galaxy S5	58
Figura 16 – Consumo de energia das estratégias de comunicação utilizadas em <i>computation offloading</i> do celular Galaxy S5	60
Figura 17 – Gráficos da comparação do consumo de energia das execuções remotas (<i>offloading</i>) do Galaxy S5	62
Figura 18 – Consumo de energia dos tipos de dados das execuções locais com o celular Galaxy S	66
Figura 19 – Comparação do consumo de energia das execuções locais e as execuções remotas (<i>offloading</i>) mais econômica no Galaxy S com relação aos tipos dos dados	67
Figura 20 – Consumo de energia das execuções remotas (<i>offloading</i>) no Galaxy S com relação aos tipos dos dados	68
Figura 21 – Consumo de energia dos tamanhos de entrada das execuções locais com o celular Galaxy S	69
Figura 22 – Comparação do consumo de energia das execuções locais e as execuções remotas (<i>offloading</i>) mais econômica no Galaxy S com relação aos tamanhos dos dados	69
Figura 23 – Consumo de energia das execuções remotas (<i>offloading</i>) no Galaxy S com relação aos tamanhos de entrada	70
Figura 24 – Consumo de energia das complexidades dos algoritmos das execuções locais com o celular Galaxy S	71
Figura 25 – Comparação do consumo de energia das execuções locais e as execuções remotas (<i>offloading</i>) mais econômica no Galaxy S com relação as complexidades assintóticas	71
Figura 26 – Consumo de energia das execuções remotas (<i>offloading</i>) no Galaxy S com relação as complexidades assintóticas	72
Figura 27 – Gráficos da comparação do consumo de energia das execuções locais e as execuções remotas (<i>offloading</i>) do Galaxy S	74
Figura 28 – Consumo de energia das estratégias de comunicação utilizadas em <i>computation offloading</i> do celular Galaxy S	75
Figura 29 – Gráficos da comparação do consumo de energia das execuções remotas (<i>offloading</i>) do Galaxy S	75

Figura 30 – Comparação do consumo de energia dos tipos de dados, tamanho e complexidade das execuções locais do Galaxy S5 vs Galaxy S	78
Figura 31 – Comparação do consumo de energia dos tipos de dado das execuções remotas do Galaxy S5 vs Galaxy S	79
Figura 32 – Comparação do consumo de energia dos tamanhos dos dados das execuções remotas do Galaxy S5 vs Galaxy S	80
Figura 33 – Comparação do consumo de energia das complexidades assintóticas das execuções remotas do Galaxy S5 vs Galaxy S	81

Lista de tabelas

Tabela 1 – Conjunto representativo de <i>smartphones</i> Android e sua evolução nos últimos anos	21
Tabela 2 – Métodos HTTP utilizados para executar serviços RESTful	35
Tabela 3 – Complexidade dos algoritmos de ordenação	43
Tabela 4 – Comparação entre o gasto de energia das execuções locais e as execuções remotas (<i>offloading</i>) que menos consumiram energia no Galaxy S5 . . .	50
Tabela 5 – Execuções em que o REST foi a alternativa mais econômica - Galaxy S5	61
Tabela 6 – Execuções em que o Socket foi a alternativa mais econômica - Galaxy S5	63
Tabela 7 – Execuções em que o gRPC foi a alternativa mais econômica - Galaxy S5	63
Tabela 8 – Execuções em que o SOAP foi a alternativa mais econômica - Galaxy S5	63
Tabela 9 – Comparação entre o gasto de energia das execuções locais contra o gasto de energia da execução remota (<i>offloading</i>) que menos consumiram energia no Galaxy S	65
Tabela 10 – Execuções remotas com REST - Galaxy S	76
Tabela 11 – Execuções remotas com Socket - Galaxy S	77
Tabela 12 – Execuções remotas com SOAP - Galaxy S	77

Lista de códigos

Código 1 – Estrutura de laço básica	24
Código 2 – Estrutura de laço recomendada	24
Código 3 – Exemplo de uma mensagem SOAP	32
Código 4 – Exemplo de uma mensagem SOAP com SOAP Fault	33
Código 5 – Exemplo do uso de HATEOAS	35
Código 6 – Estrutura de dados para utilização da serialização com <i>protocol buffers</i>	38
Código 7 – Exemplo de código instrumentado para medição do consumo de energia de um algoritmo	39

Lista de abreviaturas e siglas

API	Application programming interface
ASCII	American Standard Code for Information Interchange
CPU	Central Processing Unit
EBCDIC	Extended Binary Coded Decimal Interchange Code
HATEOAS	Hypermedia as the Engine of Application State
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LDC	Liquid Crystal Display
OLED	Organic Light-Emitting Diode
REST	Representational State Transfer
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VM	Virtual Machine
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
XML	eXtensible Markup Language
XSD	XML Schema Definition

Sumário

1	Introdução	17
1.1	<i>Motivação</i>	18
1.2	<i>Objetivo</i>	18
1.3	<i>Estrutura do documento</i>	19
2	Fundamentação teórica	20
2.1	<i>Consumo de energia em dispositivos móveis</i>	20
2.2	<i>Redução do consumo de energia em aplicativos móveis</i>	23
2.3	<i>Computation offloading</i>	25
2.4	<i>Técnicas de comunicação remota utilizadas em computation offloading</i>	29
2.4.1	Socket	29
2.4.2	Web Services	30
2.4.3	RPC - Remote Procedure Call	36
2.5	<i>Métodos de medição de consumo de energia em dispositivos móveis . .</i>	38
2.6	<i>Considerações finais</i>	40
3	Materiais e métodos	42
3.1	<i>Materiais</i>	42
3.2	<i>Métodos</i>	44
3.3	<i>Ameaças à validade</i>	46
4	Resultados e discussões	48
4.1	<i>Galaxy S5</i>	49
4.1.1	QP1: Qual a influência do tipo de dado no consumo de energia? . .	51
4.1.2	QP2: Qual a influência da quantidade de dados processados no consumo de energia?	53
4.1.3	QP3: Qual a influência da complexidade assintótica do algoritmo no consumo de energia?	55
4.1.4	QP4: Em quais situações as execuções locais consomem menos energia do que as execuções remotas?	59

4.1.5	QP5: Em quais situações as execuções remotas consomem menos energia do que as execuções locais?	59
4.1.6	QP6: Em quais situações as execuções de comunicação usadas em <i>computation offloading</i> consumiram menos energia?	59
4.2	<i>Galaxy S</i>	64
4.2.1	QP1: Qual a influência do tipo de dado no consumo de energia? . .	64
4.2.2	QP2: Qual a influência da quantidade de dados processados no consumo de energia?	68
4.2.3	QP3: Qual a influência da complexidade assintótica do algoritmo no consumo de energia?	70
4.2.4	QP4: Em quais situações as execuções locais consomem menos energia do que as execuções remotas?	72
4.2.5	QP5: Em quais situações as execuções remotas consomem menos energia do que as execuções locais?	73
4.2.6	QP6: Em quais situações as execuções de comunicação usadas em <i>computation offloading</i> consumiram menos energia?	73
4.3	<i>Comparação das análises obtidas do celular Galaxy S5 com o celular Galaxy S</i>	77
4.4	<i>Considerações finais</i>	82
5	Conclusão	85
5.1	<i>Contribuições</i>	86
5.2	<i>Trabalhos publicados</i>	87
5.3	<i>Trabalhos futuros</i>	87
	Referências	89
	Apêndice A Tabulação de todas as execuções realizadas no celular Galaxy S5	95
	Apêndice B Tabulação de todas as execuções realizadas no celular Galaxy S	96

1 Introdução

Nos últimos anos, tanto a comunidade acadêmica como a industrial têm voltado atenção ao paradigma de computação verde (BARUCHI, 2015). O termo “verde” é normalmente utilizado para referenciar práticas que reduzem o impacto no meio ambiente. A computação causa diversos tipos de impactos ambientais, incluindo o aumento do consumo de energia que pode ser diretamente responsável pelo aumento de emissão de poluentes (JUNIOR, 2014). Portanto, na comunidade de tecnologia da informação, o termo “computação verde” popularizou-se como a prática que visa à redução do consumo de energia para reduzir a poluição e a degradação ambiental na utilização de recursos computacionais (SA’ED, 2015; SILVA, 2016).

Segundo Murugesan (2008), existem quatro caminhos na computação verde:

- Uso verde: objetiva redução do consumo de energia de computadores e outros sistemas de informação;
- Descarte verde: tem o propósito de reformar e reutilizar computadores velhos e reciclar adequadamente computadores indesejados e outros equipamentos eletrônicos;
- Projeto verde: possui como finalidade projetar componentes, computadores, servidores, equipamentos de refrigeração e centros de dados energeticamente eficientes e ambientalmente saudáveis;
- Produção verde: intenta fabricar componentes eletrônicos, computadores e outros subsistemas com o mínimo de impacto para o ambiente.

O uso de dispositivos móveis aumentou consideravelmente na última década, que passaram por grandes transformações - passaram a ser dispositivos com grande capacidade, processadores multi-core, gigabytes de RAM, gigabytes de armazenamento, câmeras integradas com altas resoluções, sensores e uma grande quantidade de aplicativos complexos para executar tarefas computacionais pessoais e profissionais. A crescente popularidade da Internet das Coisas também fez do ambiente móvel alvo de muitos avanços (AAZAM; HUH, 2016). Essas mudanças fizeram com que o consumo de energia dos dispositivos móveis aumentasse consideravelmente uma vez que os usuários precisam recarregá-los cada vez mais em curtos intervalos de tempo, principalmente porque a evolução das tecnologias da bateria é historicamente lenta e não houve uma grande evolução na forma de disponibilização de energia para dispositivos móveis (TARKOMA et al., 2014).

Consequentemente, os dispositivos móveis têm sido alvo de diversas pesquisas relacionadas à computação verde. Em especial, vários estudos têm sido realizados para identificar de que forma os aplicativos móveis poderiam ser construídos para que o consumo de energia fosse menor, tanto por questões ambientais quanto para satisfazer usuários que não desejam recarregar seus dispositivos móveis em curtos intervalos de tempo. Uma das estratégias mais investigadas para reduzir o consumo de energia em aplicações móveis é conhecida como *computation offloading*, que consiste em transferir tarefas de computação específicas para uma plataforma externa, como um cluster ou uma nuvem.

1.1 Motivação

Existem vantagens e desvantagens no uso do *computation offloading*. De acordo com Kumar e Lu (2010), a energia economizada pelo descarregamento de computação depende da rede, da quantidade de computação a ser realizada e da quantidade de dados a serem transmitidos. Assim, decidir quando recorrer a essa técnica é uma preocupação fundamental para os arquitetos e desenvolvedores de sistemas móveis. As pesquisas existentes nesta área concentram-se em determinar quando utilizar o *computation offloading* pode ser vantajoso prevendo as relações entre esses três fatores.

Apesar dos trabalhos mostrarem que existe uma redução de energia com a utilização desta estratégia, não fica claro de que forma foram aplicadas, o quanto economizou-se de energia, e o que foi transferido para o servidor. Embora a tecnologia de comunicação ou o estilo arquitetônico usado para trocar dados com o serviço externo desempenhem um papel importante neste contexto, as abordagens encontradas na literatura não discutem a influência dos protocolos de comunicação e do estilo arquitetônico escolhido no consumo de energia.

1.2 Objetivo

Esta dissertação de mestrado tem por objetivo avaliar, por meio de uma pesquisa experimental, o consumo de energia das diferentes técnicas de comunicação cliente-servidor (REST, SOAP, Socket e gRPC) utilizadas em aplicativos móveis.

A contribuição esperada deste projeto de mestrado é a implementação de uma metodologia para a avaliação do comportamento do consumo de energia em dispositivos móveis e a realização de experimentos analisando as execuções locais e as execuções com diferentes tecnologias de comunicação cliente-servidor para aplicativos móveis escritos em Android, baseada em diferentes complexidades assintóticas, tamanhos de entrada e tipos de dados do algoritmo. Essas informações serão úteis para desenvolvedores e arquitetos tomarem decisões informadas sobre o uso do *computation offloading* em suas aplicações.

1.3 Estrutura do documento

O capítulo 2 deste trabalho apresenta os conceitos considerados fundamentais para a compreensão da resolução do problema abordado. Serão elucidados, portanto, aspectos os conceitos sobre o consumo de energia nos dispositivos móveis e como o computation offloadgin pode contribuir para minimizá-lo. Esses aspectos são considerados relevantes e intrinsecamente ligados, uma vez que esse trabalho visa estudar a gestão do consumo energético em dispositivos móveis, com sua consequente redução de consumo, utilizando o *computation offloading*. O capítulo 3 apresenta a metodologia e a tecnologia que será utilizada para atingir os objetivos citados no capítulo

2 Fundamentação teórica

Neste capítulo são apresentados os principais conceitos necessários para o entendimento desta pesquisa. Em um primeiro momento, disserta-se sobre o consumo de energia em dispositivos móveis; em seguida, algumas técnicas utilizadas para reduzir o consumo de energia de aplicativos móveis são apresentadas. Em especial, são apresentados os conceitos, objetivos e pesquisas relacionadas à técnica de *computation offloading*, destacando as tecnologias e arquitetura de comunicação remota possíveis neste contexto. Por fim são apresentados os métodos de medição de consumo de energia em dispositivos móveis.

2.1 Consumo de energia em dispositivos móveis

O termo *dispositivo móvel* refere-se a equipamentos eletrônicos portáteis, como, por exemplo, notebooks, telefones celulares, relógios inteligentes, dentre outros. No contexto desta dissertação, o termo *dispositivo móvel* será utilizado para designar um grupo particular dessa definição que são os *smartphones*. Um *smartphone* é uma variação do telefone celular que, além da capacidade de realizar chamadas telefônicas, oferece funcionalidades computacionais como navegação na Internet, troca de mensagens de texto, câmeras digitais, gerenciamento de informações pessoais e execução de aplicativos. Aplicativo é a denominação dada para quaisquer tipos de softwares que rodam sobre o *smartphone*. Por sua capacidade de processamento de alto desempenho, substitui, em muitos casos, o uso dos notebooks, também portáteis, principalmente em ambientes cuja a necessidade de mobilidade se faz presente (PIERER, 2016). Além de CPU de alto desempenho, os *smartphones* são providos de adequada quantidade de memória, em seus diversos níveis hierárquicos e sensores.

De acordo com Westfield e Gopalan (2016), a utilização e o aumento da complexidade dos dispositivos móveis tornaram possível o desenvolvimento de aplicativos, os quais se tornaram populares por meio de publicações em plataformas como a Play Store do Google, App Store da Apple e a Windows Phone Store, da Microsoft. Com isso, a utilização de dispositivos móveis superou a utilização de desktops para atividades diárias (ADEPU; ADLER, 2016). Como os dispositivos móveis são usados intensamente durante todo o dia para as mais variadas atividades (AGUADO, 2015), existe uma preocupação com o

consumo crescente de energia desses equipamentos, tanto por questões ambientais quanto para a satisfação de seus usuários que não desejam recarregar seus dispositivos em curtos intervalos de tempo. Portanto, essa plataforma tornou-se alvo de diversas pesquisas que visam minimizar o consumo de energia (WESTFIELD; GOPALAN, 2016).

Melhoras consideráveis foram realizadas nas CPUs nos últimos anos, um dos maiores consumidores de energia em dispositivos móveis, visando o equilíbrio entre desempenho e o consumo de energia. Um conjunto representativo de smartphones Android e sua evolução de 2009 à 2015 são apresentadas na Tabela 1. O consumo de energia reduziu e o desempenho melhorou a partir dos modelos D até o modelo A8. Em todas as cargas de trabalho reduziram de 0,8W para 0,5W, entretanto a partir do modelo S4S houve uma tendência no aumento de frequências de clock, aumentando o consumo de energia para 1,5W. Essa tendência durou por cinco gerações, sendo o pico do consumo de energia com o S5S, com 2W. Esse comportamento se mantém nos dispositivos mais recentes como o S6. O S5S possui um desempenho 10% maior que o S5Q, porém o consumo de energia aumentou quase 50% devido ao seu agressivo pipeline e subsistema de hierarquia de memória (HALPERN; ZHU; REDDI, 2016).

Tabela 1 – Conjunto representativo de *smartphones* Android e sua evolução nos últimos anos

Ano	2009	2010	2011	2012		2013		2014		2015
Fabricante	Morotola	Samsung								
Nome	Droid	Galaxy S	Nexus	Galaxy S3		Galaxy S4		Galaxy S5		Galaxy S6
Rótulo	D	S	N	S3S	S3Q	S4S	S4Q	S5S	S5Q	S6
CPU	Texas Instruments OMAP 3430	Samsung Exynos 3110	Texas Instruments OMAP 4460	Samsung Exynos 4412	Qualcomm Snapdragon MSM8960	Samsung Exynos 5410	Qualcomm Snapdragon APQ8064T	Samsung Exynos 5422	Qualcomm Snapdragon 8930AB	Samsung Exynos 7420

Fonte: (HALPERN; ZHU; REDDI, 2016)

O problema de gerenciamento de energia possui um agravante em dispositivos móveis além da CPU: sua fonte energética é baseada em bateria. A evolução das tecnologias da bateria é historicamente lenta e não houve uma grande evolução na forma de disponibilização de energia para dispositivos móveis (TARKOMA et al., 2014). A densidade das baterias de Lítio melhora cerca de 10% ao ano, o que implica que a capacidade energética da bateria é determinada por seu volume e, consequentemente, depende do tamanho do dispositivo móvel. Não se espera, portanto, que haja grande evolução na capacidade das baterias uma vez que o tamanho de dispositivos móveis como *smartphones* está começando a estabilizar em função da preferência do mercado consumidor desses aparelhos. Cerca de 98% dos usuários preferem uma tela de aproximadamente cinco polegadas (HALPERN; ZHU; REDDI, 2016).

Ao mesmo tempo que não é desejável que os dispositivos móveis tenham um tamanho maior, os usuários esperam que a cada geração mais sensores e periféricos sejam incorporados aos dispositivos móveis, e que a capacidade de processamento aumente. Consequentemente, o consumo de energia de dispositivos mais complexos aumenta (TAR-KOMA et al., 2014; HALPERN; ZHU; REDDI, 2016), enquanto a evolução da bateria está estagnada, tornando a bateria um fator limitante.

De acordo com Toh (2001), existem quatro tipos principais de baterias recarregáveis que podem ser utilizadas em dispositivos móveis:

- Níquel Cádmio (NiCd)
- Níquel-Metal Híbrido (NiMH)
- Íons-Lítio (Li-ion)
- Íon-Polímero (Li-Po)

A bateria recarregável de Níquel Cádmio é a mais antiga e por muitos anos foi a única disponível para dispositivos portáteis, ainda é possível encontrá-las em telefones sem fio e gerações antigas de celulares. Esse tipo de bateria precisa ser carregada e descarregada completamente, caso seu recarregamento seja realizado, sistematicamente, antes de sua completa descarga, sua capacidade total de carga pode ser reduzida até não ser mais possível de ser recarregada.

Para lidar com essa característica negativa das baterias Níquel Cádmio, a bateria Níquel-Metal Híbrido foi criada, mas o problema não foi extinto e sofre de elevada autodescarga, 10% a mais que a Níquel Cádmio e seu desempenho começa a ser reduzido depois de 200-300 recargas.

Surge, então, a bateria de Íons-Lítio, atualmente utilizada em dispositivos móveis. Possui amplas vantagens com relação as anteriores como sua densidade, o dobro das Níquel Cádmio, não possui problema de descarga e carregamento, sua capacidade energética não é reduzida. Está sujeita ao envelhecimento, mesmo se não for utilizada e precisa de um invólucro de metal para proteção.

A bateria de Íons-Lítio evoluiu um pouco para a versão Íon-Polímero, como esse tipo utiliza um polímero sólido seco, o invólucro de metal é desnecessário, podendo a bateria pode ser mais leve, segura e ter a espessura de um cartão de crédito (MENNEN, 2005).

A capacidade de energia da bateria pode ser descrita de diversas maneiras, como Joules por segundo ou Watts por segundo. Joule é a unidade de energia do Sistema Internacional de Unidades (SI), e é a quantidade de energia produzida por um watt por segundo (W/s) e o watt é a unidade de potência do Sistema Internacional de Unidades, que é equivalente a um joule por segundo (J/s). Os watts é a energia fornecida para o sistema operar (TOH, 2001).

2.2 Redução do consumo de energia em aplicativos móveis

As mudanças incorporadas às CPUS dos dispositivos móveis ao longo dos anos não foram suficientes para reduzir o custo de energia dos dispositivos móveis, principalmente porque as aplicações utilizadas em dispositivos móveis tornaram-se mais complexas em razão dos algoritmos utilizados, volume de dados manipulados e recursos consumidos. Por exemplo, aplicativos populares como o Facebook, o Spotify, Waze, Google Maps, Uber, Netflix, Tinder e Snapchat estão entre os aplicativos que mais consomem bateria (PATKARR, 2017; COHEN, 2017). Diversas aplicações têm recebido críticas de seus usuários com relação ao consumo de energia gerado, isso mostra que os usuários desejam o aumento da vida da bateria de seus dispositivos móveis (HALPERN; ZHU; REDDI, 2016). Portanto, faz-se necessário utilizar estratégias para reduzir o consumo de energia em nível de software (SAAB et al., 2015; WILKE et al., 2013).

O consumo de energia de software em aplicações móveis é uma preocupação crescente e, segundo Manotas et al. (2016), embora as pesquisas dessa área estejam aumentando, ainda pouco se sabe sobre as práticas e perspectivas para a redução do consumo de energia. Como diversos elementos do dispositivo móvel influenciam diretamente no consumo de energia, várias ramificações de pesquisa nessa área vem sendo desenvolvidas, estudando a influência da maneira que o aplicativo é implementado, aspectos visuais, dispositivos sensores, processamento, dentre outros.

O consumo de energia de um aplicativo pode ter influência de diversos fatores, incluindo a linguagem de programação escolhida para implementá-lo. Paul e Kundu (2010) mostram que as linguagens de programação também afetam o consumo de energia em dispositivos móveis Android. A linguagem Scala foi escolhida para ser comparada com a Java, pois ela também funciona na JVM, assim é possível executar o código em dispositivos

Android. *Benchmarks* foram realizados medindo o consumo de energia, o tempo dos benchmarks e também a utilização de memória. Os resultados mostram que Scala executa mais rápido ou mais lento do que Java, dependendo da tarefa, mas em média consome mais energia do que Java no dispositivo Android.

Práticas de implementações de código têm sido investigadas para ajudar os desenvolvedores a criarem aplicativos energeticamente eficientes, certas práticas de codificação pode reduzir o consumo de energia, como por exemplo a forma da leitura do tamanho de um vetor pode reduzir o consumo de energia. O Código 1 mostra um exemplo de laço básica e o Código 2 mostra uma boa prática na redução do consumo de energia em dispositivos móveis Android, é possível economizar 10% de energia evitando acessar uma referência ao tamanho do vetor a cada iteração do laço (LI; HALFOND, 2014).

Código 1 – Estrutura de laço básica

```

1  Object [] array;
2  for (int i=0; i<array.length; i++)
3  {
4      array[i]=null;
5  }
```

Fonte: (LI; HALFOND, 2014)

Código 2 – Estrutura de laço recomendada

```

1  Object [] array;
2  int l=array.length;
3  for (int i=0; i<l; i++)
4  {
5      array[i]=null;
6  }
```

Fonte: (LI; HALFOND, 2014)

Não só as estruturas internas de um aplicativo podem influenciar no consumo de energia, mas também elementos da interface gráfica. A cor escolhida para o aplicativo tem impacto no consumo de energia, que pode ser estimado usando os componentes de cor de cada pixel. A técnica elaborada por Linares-Vásquez et al. (2015) gera composições de cores que reduzem o consumo de energia por meio do aumento do contraste da cor original do aplicativo. O estudo realizado com 25 aplicativos Android mostra que é possível reduzir cerca de um terço do consumo de energia com essa técnica.

Em relação ao uso de recursos do dispositivo móvel, testes realizados em elementos internos de um dispositivo móvel mostraram que o GPS é o que mais consome energia, se-

guido pelo Bluetooth e pelo Wi-Fi. A operação de detecção de localização por GPS consome muita energia, portanto algoritmos mais eficientes e novas arquiteturas têm sido propostas, como, por exemplo, o uso de estimativas probabilísticas e determinísticas (BERNAL et al., 2010).

O uso de recursos internos, incluindo memória e processador, tem grande impacto no consumo de energia. Portanto, uma técnica comumente explorada para a redução do consumo de energia em dispositivos móveis é o uso de *computation offloading*, a qual tem por objetivo delegar a execução de tarefas para servidores remotos, os quais possuem mais recursos computacionais. Como este projeto de pesquisa contribui com o uso desta técnica específica, mais detalhes sobre ela são apresentados na próxima seção.

2.3 *Computation offloading*

Computation offloading consiste em transferir a execução de tarefas para um servidor remoto por diversos motivos (DE, 2016), como, por exemplo, minimizar o consumo de energia, aumentar desempenho ou minimizar a utilização de recursos locais. As decisões sobre executar uma tarefa localmente ou em um servidor remoto podem variar de acordo com os objetivos estabelecidos. Conforme exemplificado na Figura 1, o custo benefício da aplicação do *computation offloading* deve ponderar as prioridades do usuário, os recursos de *hardware* disponíveis e necessários.

Quando o objetivo do *offloading* é apenas reduzir o consumo de bateria, deve-se ponderar se o custo energético de executar as tarefas localmente é superior ao custo de executá-las remotamente, pois, neste caso, existe um custo extra de comunicação com a rede que pode ser superior ao custo de execução local. Alguns fatores podem afetar este cenário, como a quantidade de dados a ser processada, capacidade de processamento do dispositivo e a quantidade de pacotes enviados e recebidos através da rede (SAAB; CHEHAB; KAYSSI, 2013; KUMAR; LU, 2010), o tipo de rede utilizado, a complexidade da aplicação (AKHERFI; GERNDT; HARROUD, 2016), a quantidade de memória, bateria e capacidade da CPU (KUMAR et al., 2013; FAHIM; MTIBAA; HARRAS, 2013).

Experimentos mostraram que realizar o *computation offloading* de todo o aplicativo pode aumentar o consumo de energia, sendo preferível executar tarefas simples e com pouca carga computacional no próprio dispositivo móvel, para que o custo de comunicação

Figura 1 – Análise de custo benefício e tomada de decisão para *computation offloading*

Fonte: Adaptado de (FERNANDO; LOKE; RAHAYU, 2013)

com a rede não seja maior do que o custo de execução local (SAAB; CHEHAB; KAYSSI, 2013), (CORRAL et al., 2015), (PATRA; ROY; CHOWDHURY, 2015).

Algumas restrições também podem existir com relação ao uso do *computation offloading* de método dos aplicativos, por acessarem rotinas do sistema operacional, por necessitarem da realização do sincronismo de dados ou até mesmo a necessidade de refatoração de métodos do aplicativo, principalmente por existir dependência entre métodos (SAARINEN et al., 2012).

Um dos fatores limitantes para o uso de *computation offloading* é o fato do aplicativo precisar estar conectado a uma rede o tempo todo. Entretanto, algumas abordagens têm sido exploradas para realizar o *computation offloading* com o intuito de mitigar este problema quando o aplicativo está impossibilitado de se comunicar com a rede. Por exemplo, é possível fazer uma réplica eficiente de informações localmente para tornar possível o sincronismo entre execuções locais e remotas, podendo dessa forma evitar a inutilização do aplicativo por não depender exclusivamente da conexão com a rede (KWON; TILEVICH, 2012).

Técnicas de otimização de envio de pacotes pela rede enviando os mais urgentes e acumulando os menos prioritários; algoritmo para troca dinâmica da rede 3G para a

rede Wi-Fi, podendo ser configurável para, por exemplo, realizar a troca de acordo com o quantidade de informações que se quer enviar e melhorias no consumo de energia por meio da redução da espera do carregamento de uma página web (LIN et al., 2010), são exemplos de outros estudos realizados.

Atualmente, a nuvem é o ambiente mais popular para os aplicativos móveis executarem tarefas utilizando recursos remotos, principalmente por sua alta disponibilidade, baixo custo e alto poder computacional (LI et al., 2017). A integração entre o dispositivo móvel e serviços em nuvem é conhecido por *Mobile Cloud Computing* (DE, 2016).

A principal característica da computação em nuvem é a elasticidade, pois permite que os recursos sejam utilizados na quantidade que forem necessários, aumentando ou diminuindo a capacidade computacional de forma dinâmica (TAURION, 2009). A ideia da computação em nuvem é manter dados e aplicações disponibilizadas na Internet sem a necessidade de instalações ou armazenamento de dados localmente. Desse modo, computação em nuvem oferece a possibilidade de acessar arquivos, dados e aplicações de praticamente qualquer lugar e a qualquer momento, utilizando a Internet com o computador, o browser ou dispositivos móveis (GOMES; FERTIG, 2016).

De acordo com Fernando, Loke e Rahayu (2013), os principais aspectos do uso da computação em nuvem como apoio aos dispositivos móveis, em função da sua limitação de recursos computacionais, estão sendo abordados em:

- Nível de usuário final: para incentivar a participação do compartilhamento de seus recursos móveis no processamento de uma tarefa comum. Se muitos usuários precisam executar uma mesma tarefa, esta pode ser particionada para que cada usuário faça o processamento de uma fração.
- Nível de serviço e aplicativo: os aplicativos podem se conectar e solicitar serviços hospedados em uma nuvem remota por meio de interfaces. No entanto, é necessário considerar restrições como perda frequente de conectividade, baixa disponibilidade de recursos computacionais e baixa largura de banda.
- Nível de privacidade, segurança e confiabilidade: existem riscos de segurança que os usuários precisam considerar na computação em nuvem, pois o descarregamento de dados para a nuvem pode significar perder o controle sobre os dados. Assim, auditorias e certificações de segurança devem ser exigidas dos prestadores dos serviços na nuvem. Alguns aspectos relacionados ao armazenamento de dados na nuvem

são muito importantes neste contexto: é importante que os dados de cada usuário estejam separados de outros por criptografias eficientes; os prestadores de serviços em nuvem devem fornecer recuperação de dados e serviços em caso de falha técnica ou outro desastre; é preciso haver suporte de pesquisa a atividades inapropriadas ou ilegais; e uma garantia é necessária de que os dados dos usuários estejam seguros e acessíveis mesmo se a empresa que fornece o serviço em nuvem falir. Além disso, deve existir um sistema de autenticação entre o dispositivo móvel e a nuvem. Com relação à privacidade, os usuários precisam concordar e saber quais informações pessoais podem ser visíveis ao público e poder ter controle sobre seus dados.

- Nível de gestão de dados: além da sensível questão na segurança, a acessibilidade e disponibilidade dos dados é um desafio, uma vez que a falta de recursos, como conectividade de rede, podem inviabilizar o acesso. A limitação de armazenamento do dispositivo móvel faz com que as transações de recebimento dos dados tenham que ser tão leves que não sobrecarreguem o dispositivo, tampouco seu banco de dados.
- Nível operacional: as questões operacionais referem-se, principalmente, a questões tecnológicas como o uso de tecnologias para viabilizar o *computation offloading* e análise de seu custo-benefício. Existem três principais métodos de realizar o *offloading*: métodos de comunicação Cliente-Servidor, virtualização e agentes móveis. O método de comunicação cliente-servidor é feito utilizando tecnologias como RPC e Socket, por exemplo. Para utilizar esse método é necessário que o aplicativo seja desenvolvido considerando cada tipo de comunicação. O método de virtualização gera uma cópia do conteúdo que será processado no dispositivo móvel na memória do servidor. Apesar de não prever alterações de código, a migração da VM (máquina virtual) é demorada e a carga de trabalho pode ser computacionalmente custosa para dispositivos móveis. O último método prevê um escalonador de custo de processamento no dispositivo móvel, distribuindo *jobs* para o servidor quando o escalonador considerar a carga alta para o dispositivo. A manutenção de um escalonador em tempo real pode tornar o método oneroso para o dispositivo móvel.

Apesar dos diferentes aspectos a se considerar ao aplicar a técnica de *computation offloading*, e da complexidade da aplicação aumentar em razão da implementação de acessos remotos, a existência da necessidade de diminuir o gasto energético demanda o uso desta técnica (CORRAL et al., 2015). Como a comunicação com a rede é um aspecto

importante neste contexto, a seguir são discutidas as técnicas de comunicação remotas mais comumente adotadas em *computation offloading*.

2.4 Técnicas de comunicação remota utilizadas em *computation offloading*

De acordo com pesquisas exploratórias, chamadas externas em ambiente Android podem ser realizadas por meio de Socket, *Web Services* ou RPC.

2.4.1 Socket

Socket é uma interface de entrada e saída de processos, que enviam mensagens por esse canal que tem acesso aos protocolos de transporte da rede (OLIVEIRA; FRAGA; MONTEZ, 2002) e é o modelo cliente/servidor mais utilizado para sistemas e aplicações distribuídos, tendo a seguinte dinâmica (NETO; SILVA, 2008):

1. Cliente envia mensagem para o servidor solicitando tarefa;
2. Servidor executa a tarefa;
3. Servidor envia resposta para o cliente.

Alguns exemplos de operações feitas via Socket são como a conexão entre máquinas, envio e recebimento de dados, aguardar conexões em portas específicas ou até mesmo encerrar conexões (NETO; SILVA, 2008). Existe uma infindável gama de possibilidades para desenvolvimento de sistemas distribuídos baseados em Socket, no entanto, esta tecnologia também tem dificuldades para ser implementada, dificuldades essas que não podem ser ignoradas, como heterogeneidade de representação de dados ou a falta de transparência de distribuição, e podem aumentar a complexidade do desenvolvimento do sistema (NETO; SILVA, 2008). Considere, por exemplo, a representação de caracteres codificados em padrão ASCII sendo empacotados em um computador com um tipo de CPU e desempacotado em outro computador com outro tipo de CPU. Para minimizar esse tipo de problemas, deve-se criar um formato padrão e trabalhar apenas sobre ele ou passar, conjuntamente com os dados, o formato adotado (OLIVEIRA; FRAGA; MONTEZ, 2002).

Uma forma de utilização do Socket, como a utilizada na linguagem de programação Java, é o modo orientado à conexão, sobre o protocolo TCP (*Transmission Control Protocol*). Outra forma é que ocorre sobre o protocolo UDP (*User Datagram Protocol*),

denominada orientado à datagrama. Ambos os modos funcionam sobre o protocolo IP (Internet Protocol) (NETO; SILVA, 2008). A Figura 2 apresenta uma representação esquemática de aplicação distribuída baseada em Socket.

Figura 2 – Representação esquemática de aplicação distribuída baseada em Socket

Fonte: Adaptado de (GOMES, 2014)

No modo orientado à conexão, o servidor escolhe uma porta e aguarda conexões que cheguem por ela. O cliente deve conhecer a porta que está escutando no servidor e faz uma requisição por ela. Se não houver nenhum erro ou exceção, o servidor aceita a conexão e gera-se um Socket, criando, portanto, um canal de comunicação cliente/servidor. O servidor fica repetidamente aguardando conexões e gerando sockets a cada solicitação de cliente (NETO; SILVA, 2008).

No modo orientado a datagrama, o protocolo UDP, por não prestar um serviço orientado à conexão, portanto sem garantia de entrega, demanda que as aplicações tenham garantias de recuperação de dados perdidos e eliminação de duplicidades (ALBUQUERQUE, 2001). Apesar de menos confiável, a implementação é simplificada e torna as conexões Socket mais rápidas. Basicamente, neste modo, existe apenas trocas de mensagens, que se trata de um datagrama que vai de um remetente para um receptor que, se não estiver aguardando, a mensagem é perdida.

2.4.2 Web Services

Uma das formas de realizar chamadas externas em ambiente Android é por meio de *Web Services*. *Web Service* é uma solução cliente-servidor que tem a capacidade de disponibilizar um recurso na Internet, tendo como principal vantagem a possibilidade de

comunicação entre diferentes aplicações que podem ter sido desenvolvidas em diferentes plataformas (SNELL; TIDWELL; KULCHENKO, 2001).

O *Web Service* é capaz de enviar e receber mensagens usando algumas combinações de protocolos padrões da Internet, chamando procedimentos que rodam em um servidor. A mensagem de requisição para esse servidor contém a especificação de qual serviço executar e os parâmetros necessários. O servidor, por sua vez, responde à requisição solicitada com os resultados da execução (SNELL; TIDWELL; KULCHENKO, 2001).

Atualmente, existem duas tecnologias que disponibilizam um *Web Service*: SOAP (*Simple Object Access Protocol*) e REST (*REpresentational State Transfer*), os quais são detalhadas a seguir.

SOAP - *Simple Object Access Protocol*

SOAP é um protocolo que utiliza o formato XML para a troca de dados entre pontos de um ambiente descentralizado e, normalmente, é realizada por meio de HTTP (BOX et al., 2000). SOAP é constituído por três partes:

- O envelope define uma estrutura geral para expressar o que está em uma mensagem e o que em sua composição é opcional ou obrigatório;
- As regras de codificação que definem um mecanismo de serialização a ser utilizado nas trocas de dados;
- A representação RPC que define as convenções de chamadas e respostas remotas.

O processamento de uma mensagem SOAP por uma aplicação deve identificar todos as partes desta mensagem, verificar se todas as partes obrigatórias estão presentes e realizar seu processamento, descartando-a caso contrário (partes opcionais podem ser ignoradas). O Código 3 é um exemplo de uma mensagem SOAP. Apesar de ser fundamentalmente unidirecional, uma mensagem SOAP é frequentemente combinada para implementar padrões do tipo request/response (BOX et al., 2000).

Uma mensagem SOAP consiste em elementos chamados: envelope, cabeçalho e corpo (OLIVEIRA, 2012), conforme Figura 3. O envelope engloba os elementos cabeçalho, que é opcional e só pode existir uma única ocorrência desse elemento na mensagem, e o elemento corpo que é obrigatório e também só pode existir um única ocorrência desse elemento (BOX et al., 2000).

Código 3 – Exemplo de uma mensagem SOAP

```

1      <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap-envelope/"
2      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
3      <SOAP-ENV:Header>
4          <t:Transaction xmlns:t="some-URI" SOAP-ENV:mustUnderstand="1">
5              5
6          </t:Transaction>
7      </SOAP-ENV:Header>
8      <SOAP-ENV:Body>
9          <m:GetLastTradePrice xmlns:m="Some-URI">
10             <symbol>DEF</symbol>
11          </m:GetLastTradePrice>
12      </SOAP-ENV:Body>
13  </SOAP-ENV:Envelope>

```

Fonte: (BOX et al., 2000)

O propósito do cabeçalho é comunicar informação contextual relevante para o processamento da mensagem SOAP, como, por exemplo, credenciais de autenticação. Seus elementos filhos devem trazer a identificação do elemento e podem trazer a codificação dos dados e seus atributos (BOX et al., 2000).

Figura 3 – Representação de uma mensagem SOAP, composta pelos elementos: envelope, cabeçalho e corpo

Fonte: Adaptado de (OLIVEIRA, 2012)

O corpo, conforme Box et al. (2000), é a maneira de codificação dos dados que serão enviados para o sistema computacional que o receberá. Trata-se de um elemento

filho do envelope e quando este traz o elemento cabeçalho, o corpo deve obrigatoriamente vir logo na sequência. Assim como no cabeçalho, as entradas do corpo devem conter um identificador e podem trazer a codificação dos dados.

O corpo contém um elemento, de única instância, que é usado em caso de ocorrência de erros, denominado *fault* que contém o código de erro e uma sequência de caracteres que objetiva a explicação do erro em alto nível. Um exemplo do SOAP *fault* pode ser encontrado no Código 4.

Código 4 – Exemplo de uma mensagem SOAP com SOAP Fault

```

1      HTTP/1.1 500 Internal Server Error
2      Content-Type: text/xml; charset="utf-8"
3      Content-Length: nnnn
4      <SOAP-ENV:Envelope
5      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
6      <SOAP-ENV:Body>
7      <SOAP-ENV:Fault>
8      <faultcode>SOAP-ENV:MustUnderstand</faultcode>
9      <faultstring>SOAP Must Understand Error</faultstring>
10     </SOAP-ENV:Fault>
11   </SOAP-ENV:Body>
12 </SOAP-ENV:Envelope>

```

Fonte: (BOX et al., 2000)

Para facilitar a integração entre o cliente e o serviço, a W3C mantém o formato WSDL (Web Service Description Language) para padronizar as descrições das funcionalidades oferecidas de forma independente de plataforma ou linguagem, informando, por exemplo, quais os parâmetros, tipos de dados que são esperados e qual o tipo de dado de retorno. Pode ser definido como um documento que descreve uma interface, geralmente associada a um *Web Service* SOAP, provendo aos usuários um ponto de contato. Um documento WSDL completo contém uma descrição referente ao *Web Service* e os detalhes necessários que o usuário deve seguir para consumir o serviço. Normalmente um documento WSDL é construído utilizando-se *XML Schema Definition* (XSD) (CURBERA et al., 2002).

REST - *REpresentational State Transfer*

Web Service REST está ganhando popularidade, principalmente por ser mais fácil de implementar e utilizar que SOAP (MANGLER; BERAN; SCHIKUTA, 2010). REST é um estilo arquitetural, proposto por Roy Fielding, que consiste em uma série de princípios

e restrições. Quando um serviço segue os princípios propostos, em termos gerais, quer dizer que é um *Web Service* RESTful (DEWAILLY, 2015).

Fielding (2000), em sua tese de doutorado, definiu os seguintes princípios:

- **Cliente-servidor:** Este princípio significa que a implementação deve seguir um padrão cliente e servidor, pois tem como vantagem em o cliente e o servidor poderem ser implementados independentemente, além do cliente não precisar se preocupar com a persistência dos dados, banco de dados, escalabilidade, por exemplo;
- **Não manter estado:** cada requisição realizada pelo cliente deve conter todas as informações necessárias para o servidor. Deste modo, é possível facilmente escalar os servidores horizontalmente somente adicionando mais instancia de máquinas;
- **Cacheável:** é permitido o cliente fazer cache das respostas do servidor. O cliente indica ao servidor para qual resposta o cache deve ser feito e por quanto tempo. Esse princípio tem como objetivo diminuir as interações entre o cliente e o servidor, melhorando o desempenho e largura de banda;
- **Sistemas em camadas:** este princípio descreve que é possível criar camadas entre o cliente e o servidor, tendo cada camada sua própria funcionalidade. Por exemplo, pode existir um firewall, um balanceador de carga, um *proxy* reverso, entre outras funcionalidades;
- **Interface uniforme:** este princípio define o contrato entre o cliente e o servidor, tendo quatro principais aspectos (DIRKSEN, 2015):
 - Cada recurso deve ter um identificador único, geralmente isto é realizado por meio de uma URI e sua representação mais comum é feita utilizando JSON;
 - O cliente pode modificar um recurso, atualizando sua representação e enviando para o servidor;
 - O cliente não precisa ter nenhum conhecimento prévio para processar as mensagens que devem apresentar tudo o que for necessário para o seu processamento;
 - Ter um conjunto de operações padronizadas, como GET, POST, DELETE, PUT, que formam as quatro operações básicas, como mostra a Tabela 2 (PURUSHOTHAMAN, 2015);
 - HATEOAS (*Hypermedia as the Engine of Application State*) é um dos principais aspectos do REST, pois visa associar elementos de dados a um controle de hipermídia. Um exemplo de controle hipermídia é uma representação de recursos

Tabela 2 – Métodos HTTP utilizados para executar serviços RESTful

Ação	Equivalente
Criação	POST
Consulta	GET
Atualização	PUT
Exclusão	DELETE

Fonte: (PURUSHOTHAMAN, 2015)

com um hiperlink para outro recurso. Como no Código 5, não só os itens e os preços são apresentados, mas a URI para cada recurso é mostrado, fornecendo informações claras sobre como acessar esses recursos (SPRING, 2017).

Código 5 – Exemplo do uso de HATEOAS

```

1  {
2      "content": [ {
3          "price": 499.00,
4          "description": "Apple_tablet_device",
5          "name": "iPad",
6          "links": [ {
7              "rel": "self",
8              "href": "http://localhost:8080/product/1"
9          } ],
10         "attributes": {
11             "connector": "socket"
12         }
13     }, {
14         "price": 49.00,
15         "description": "Dock_for_iPhone/iPad",
16         "name": "Dock",
17         "links": [ {
18             "rel": "self",
19             "href": "http://localhost:8080/product/3"
20         } ],
21         "attributes": {
22             "connector": "plug"
23         }
24     } ],
25     "links": [ {
26         "rel": "product.search",
27         "href": "http://localhost:8080/product/search"
28     } ]
29 }
```

Fonte: (SPRING, 2017)

2.4.3 RPC - *Remote Procedure Call*

Em 1984, Birrel e Nelson, sugeriram uma forma de comunicação que permitisse que programas em uma máquina chamassem procedimentos em outras máquinas sem que a troca de mensagens fosse visível para o programador. Essa troca ocorreria da seguinte maneira: quando um processo em uma máquina A chamasse um procedimento na máquina B, o processo chamador em A seria suspenso e o procedimento na máquina B executado. Entretanto, como o procedimento chamado está localizado em outra máquina, ou seja, em espaços de memória diferentes, é necessário passar parâmetros e resultados para ser possível a realização dessa proposta de comunicação. Além disso, as máquinas podem possuir sistemas operacionais diferentes que podem tratar a representação de dados de formas diferentes. Por exemplo, mainframes IBM usam o código de caracteres EBCDIC, enquanto computadores pessoais IBM usam ASCII. Para lidar com esse problema, é realizada uma conversão de dados para um formato comum que é conhecido pelas duas máquinas - essa conversão é chamada de *marshalling* e *unmarshalling*, quando os dados são desconvertidos para um formato específico utilizado na outra máquina (TANENBAUM; STEEN et al., 2008).

A comunicação entre dois computadores, utilizando RPC, geralmente é feita de forma síncrona (FUKS; PIMENTEL, 2011), mas o RPC também suporta mensagens assíncronas quando necessário (TANENBAUM; STEEN et al., 2008), utilizadas em contextos como transferência de dinheiro de uma conta para outra ou adicionar dados em um banco de dados.

Quando se é necessário chamar um método em um objeto remoto, é utilizado o recurso RMI (*Remote Method Invocation*), semelhante ao RPC para montagem e transferência de parâmetros, mas com a RMI é possível passar objetos como parâmetros aos métodos remotos enquanto com RPC os dados são passados a um procedimento remoto na forma de uma estrutura de dados comum (SILBERSCHATZ, 2008).

A RMI provê as funcionalidades de uma plataforma de objetos distribuídos para plataformas como a linguagem de programação Java. No contexto dessa linguagem, é possível que um objeto em uma máquina virtual Java possa interagir remotamente com objetos de outras máquinas virtuais Java independentemente de sua localização, utilizando serialização e deserialização de objetos (ORACLE, 2016). A estrutura do RMI, como

mostrada na Figura 4, consiste em camadas de *stubs* e *skeletons*, referências remotas e transporte (NIEMEYER; KNUDSEN, 2005; GROUP et al., 2007), as quais são detalhadas a seguir:

- Camada *stubs* e *skeletons*: os stubs recebem os parâmetros dos métodos exportados pelo objeto reencaminhando-os para o lado do servidor o qual serão interpretados por uma instância de uma classe skeleton. O skeleton recebe os parâmetros enviados pelo *stub* e executa as respectivas chamadas no objeto remoto;
- Camada de referências remotas: é o *middleware* entre a camada de *stub/skeleton* e o protocolo de transporte. É nesta camada que são criadas e gerenciadas as referências remotas aos objetos;
- Camada de transporte: realiza a comunicação entre as várias máquinas virtuais Java por TCP/IP.

Figura 4 – Representação da estrutura arquitetural do RMI composta por *stubs* e *skeletons*, referências remotas e transporte

Fonte: Adaptado de (GROUP et al., 2007)

gRPC é um exemplo de implementação do RPC realizada pelo Google ®, que interage com servidores e clientes de diferentes linguagens de programação. Assim, pode-se, por exemplo, criar um servidor gRPC em Java com clientes implementados em linguagem Android, Go, Python ou Ruby (GOOGLE, 2017).

No gRPC, um aplicativo cliente pode chamar diretamente métodos com seus parâmetros e tipos de retorno em um servidor como se fosse uma chamada local. Por padrão, o gRPC usa *protocol buffers*, que consiste em um mecanismo linguístico neutro, neutro em plataforma e neutro para serializar dados estruturados. Para utilizar os *protocol buffers* é necessário definir a estrutura para os dados que deseja-se serializar, como no Código 6 que apresenta um exemplo simples deste tipo de estrutura. Os dados do *protocol buffer* são estruturados como mensagens, em que cada mensagem é um pequeno registro lógico de informações contendo uma série de pares nome-valor chamados campos. O gRPC realiza *streaming* desses dados, assim, quando o cliente envia uma solicitação ao servidor e recebe um *stream* de dados como retorno, o cliente lê do *stream* retornado até que não haja mais mensagens, o mesmo acontece com o servidor, quando ele recebe um *stream* de dados do cliente (GOOGLE, 2017).

Código 6 – Estrutura de dados para utilização da serialização com *protocol buffers*

```

1      message Person {
2          string name = 1;
3          int32 id = 2;
4          bool has_ponycopter = 3;
5      }
```

Fonte: (GOOGLE, 2017)

2.5 Métodos de medição de consumo de energia em dispositivos móveis

Depois da realização de uma pesquisa exploratória, foi possível perceber que existem três métodos mais comumente utilizados para analisar o consumo de energia em dispositivos móveis: medição do consumo de energia gerado pelos componentes de *hardware*, medição interna e medição via *hardware*, utilizando um multímetro. No primeiro método, todos os principais componentes de *hardware* e suas arquiteturas são considerados no impacto como, por exemplo, a CPU e seus percentuais de utilização (FISCHER; BRISOLARA; MATTOS, 2015).

O PowerTutor é um aplicativo Android que utiliza esse método estimando o consumo dos principais componentes de *hardware* de aplicativos em execução. Existem fatores que influenciam o consumo de energia em um componente de *hardware*, como, por exemplo, o consumo de energia da CPU é dado por meio de sua utilização e seu nível de frequência, as telas OLED / LCD por seus níveis de brilho. De acordo com Fischer, Brisolara e Mattos

(2015) e Yang (2012), os principais fatores que influenciam o consumo de energia dos seguintes componentes de *hardware* são:

- CPU: utilização da CPU e nível de frequência;
- OLED / LCD: nível de brilho e informações de pixel;
- Wi-Fi: pacotes transmitidos por segundo;
- 3G: pacotes transmitidos por segundo;
- GPS: Número de satélites detectados e os estados de energia do dispositivo GPS (ativo, hibernação, desligado);
- Áudio: estados de energia do dispositivo de áudio (ligado, desligado).

A interface gráfica do PowerTutor oferece uma forma de iniciar e parar a medição do consumo de energia. Ela também exibe uma lista de aplicativos ordenada pela quantidade de energia consumida. Também é possível ver, em tempo real, a energia consumida pelo aplicativo e pelos seus componentes (YANG, 2012).

Um outro software que realiza a medição da energia consumida pelos componentes de *hardware* é o AppScope, que consiste em um sistema que monitora o consumo de energia do *hardware* de um aplicativo no nível do *kernel* (HAMZAOUI et al., 2014). Por meio de uma aplicação desktop Java disponibilizada, é possível acompanhar em tempo real o consumo de energia dos *hardwares*, para isso é necessário conectar o dispositivo móvel ao computador. Os componentes de *hardware* para os quais o AppScope realiza a medição de energia são praticamente os mesmos do Power Tutor (CPU, LCD, Wi-Fi, 3G e GPS), com exceção dos componentes de áudio.

O método de medição interna de energia utiliza amostras de tensão e corrente extraídas a partir das informações de bateria disponibilizadas por meio de interfaces pelo sistema operacional. Diferentemente da abordagem anterior, que só faz medições considerando o aplicativo como um todo, esta abordagem é capaz de medir a energia de partes do código do aplicativo em tempo de execução. Por exemplo, em dispositivos Android, o estado da bateria é acessível, assim o código pode ser instrumentado para recuperar o estado inicial e final da bateria após a execução de um algoritmo. Essa lógica pode ser melhor visualizada na Código 7, na qual somente a energia gasta pelo algoritmo Bubble Sort será medida (FISCHER; BRISOLARA; MATTOS, 2015).

Código 7 – Exemplo de código instrumentado para medição do consumo de energia de um algoritmo

```

1      Context context = getBaseContext();
2      EnergyMonitor energy = new EnergyMonitor();
3      energy.start();
4      bubbleSort(array);
5      energy.stop();

```

Fonte: (FISCHER; BRISOLARA; MATTOS, 2015)

A interface que disponibiliza acesso ao estado da bateria, no sistema operacional Android, é a `BatteryManager`, a qual fornece maneiras de obter informações sobre o nível de carga da bateria (KAPETANAKIS; PANAGIOTAKIS, 2012).

Para realizar a medição de forma externa, utilizando um *hardware*, é necessário instalar um multímetro na bateria em circuito paralelo, usando as pontas do aparelho nos pólos da bateria, assim é possível capturar a energia gasta da execução de uma aplicação, pois a bateria possui uma corrente elétrica contínua (KAPETANAKIS; PANAGIOTAKIS, 2012).

2.6 Considerações finais

O avanço tecnológico dos dispositivos móveis e sua utilização incorreu no aumento do consumo de energia. Por outro lado, a bateria não acompanhou esse avanço, limitando dessa forma a autonomia dos dispositivos móveis no que se refere à disponibilidade de energia. Este contexto específico tornou-se importante de tal forma que estudos têm sido realizados para identificar como esse consumo pode ser reduzido. Dentre as propostas de redução de energia está a técnica de *computation offloading*, que tem sido utilizado visando reduzir o consumo de energia e proporcionar uma maior capacidade de processamento sem estar diretamente limitado pela bateria e pelo processamento do dispositivo, usando principalmente recursos de computação em nuvem.

Apesar de muitas abordagens terem sido publicadas sobre a técnica de *computation offloading* (SAAB; CHEHAB; KAYSSI, 2013; KUMAR; LU, 2010; AKHERFI; GERNDT; HARROUD, 2016; FAHIM; MTIBAA; HARRAS, 2013; SAARINEN et al., 2012; KWON; TILEVICH, 2013), não foram encontrados estudos que considerassem a influência da forma de comunicação com a rede. Pesquisas já mostraram que a comunicação com a rede, como o uso de Wifi e 3G/4G, são fatores que tem grande influência no consumo de energia (GENG

et al., 2015). Desta forma, percebe-se esta lacuna nas pesquisas realizadas na área. Acredita-se que entender as vantagens e desvantagens no uso de diferentes tecnologias e arquiteturas de comunicação são fundamentais para extrair ao máximo as vantagens da técnica de *computation offloading*.

3 Materiais e métodos

Com a utilização da técnica de *computation offloading* é possível transferir a responsabilidade do processamento de uma operação para um servidor por diversos motivos, incluindo a redução no consumo de energia no dispositivo móvel. Entretanto, é preciso entender em que situações a utilização do *computation offloading* é recomendável, pois existem outros fatores que podem influenciar o consumo de energia.

Em geral, o custo do consumo de energia da comunicação com a rede para a execução de tarefas entre o dispositivo móvel e o servidor tem grande influência nesta decisão, e está relacionado com três fatores: rede, quantidade de computação e quantidade de dados (KUMAR; LU, 2010). É sabido que esses fatores, individualmente, podem influenciar no consumo energético do dispositivo móvel, no entanto, seu efeito agregado também pode causar influência. Portanto, a seguir são apresentados detalhes de experimentos realizados para mostrar a influência da estratégia de comunicação com a rede, de algoritmos de diferentes complexidades, e de diferentes tipos e quantidade de dados a serem processados. Essas análises podem fornecer parâmetros estáticos para desenvolvedores de software e arquitetos para tomar decisões informadas sobre quando utilizar processamento local ou remoto, de acordo com as características de suas aplicações.

3.1 Materiais

Foi realizada uma investigação sobre os possíveis meios de comunicação entre o cliente e um servidor no contexto de aplicações Android. O sistema operacional Android foi escolhido por possuir uma extensa comunidade *open source* e ferramentas de desenvolvimento gratuitas. As técnicas que podem ser aplicadas em dispositivos móveis para realizar um processamento remoto são: Socket, RPC, SOAP e REST. O lado do servidor foi implementado em linguagem Java, e para a comunicação com o cliente, nesse caso um aplicativo Android, foi utilizado Apache Axis para disponibilização dos algoritmos por meio de *Web Services* SOAP e a biblioteca Jersey para a disponibilização de *Web Services* REST. A implementação do Socket foi feita com a biblioteca da própria linguagem de programação Java e para o RPC foi utilizada a implementação do Google ®, chamada gRPC.

Para o consumo dos *Web Services* SOAP do lado do cliente, o aplicativo Android, foi utilizada a biblioteca KSoap; e para os *Web Services* REST, a própria classe de conexão HTTP do Android, assim como classe nativa Android para o cliente Socket. No caso do gRPC, foi feita uma adaptação de cliente disponibilizado na própria documentação da API para Android.

Algoritmos de ordenação foram escolhidos para representar diferentes quantidades de computação nas tarefas a serem executadas nos aplicativos móveis. Este tipo de algoritmo foi escolhido principalmente por ser constantemente utilizado no desenvolvimento de qualquer software e também por suas complexidades assintóticas serem conhecidas. Em aplicativos, por exemplo, são utilizados para ordenar dados para melhor exibição como nomes por ordem alfabética, cidades em ordem de população, vendas por preço, dentre outros (BUNSE et al., 2009). A Tabela 3 mostra o conjunto de algoritmos de ordenação selecionado. Foram selecionados os algoritmos clássicos de ordenação como Bubble Sort e Selection Sort, considerados algoritmos simples; e o Heap Sort, considerado mais sofisticado (MCMILLAN, 2007).

Tabela 3 – Complexidade dos algoritmos de ordenação

Algoritmo	Complexidade Melhor Caso	Complexidade Pior Caso
Bubble Sort	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
Heap Sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$
Selection Sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$

Fonte: Adaptado de (MCMILLAN, 2007)

Como a quantidade de dados processados pelos algoritmos e transmitidos pela rede também influenciam o consumo de energia, decidiu-se utilizar vetores com diferentes tamanhos e tipos de dados para serem ordenados pelos algoritmos de ordenação. Assim, ficou definido os tamanhos de vetores em 1.000, 10.000 e 100.000 elementos, com tipos de dados inteiros, floats e objects. Para Object foi criado um tipo padrão *Java Bean* contendo três atributos: um de tipo inteiro e dois de tipo String.

Diversas são as formas propostas na literatura para a medição do consumo de energia de aplicativos móveis. Dentre as alternativas já apresentadas no Capítulo 2, a interface BatteryManager disponibilizada pelo Android foi escolhida para ser utilizada para medir a energia consumida dos experimentos realizados, sendo possível, assim, automatizar a execução dos experimentos.

Dessa forma, um aplicativo Android nativo foi criado para executar os algoritmos apresentados na Tabela 3. Os algoritmos recebem vetores com valores do tipo inteiro de tamanhos variados escolhidos pelo usuário e são executados tanto local quanto remotamente. O aplicativo gerado permite escolher se os algoritmos serão executados considerando o pior caso ou o melhor caso. Quando o melhor caso é selecionado, o vetor a ser ordenado é construído de forma ordenada, começando de 1 e recebendo incrementos de uma unidade a cada nova posição criada. Quando o pior caso é selecionado, o vetor é construído de forma desordenada de tal forma que a primeira posição recebe o valor equivalente ao número de elementos do vetor e a cada nova posição este valor é decrementado em uma unidade, de tal forma que o vetor fique ordenado de forma decrescente.

O servidor utilizado para hospedar os serviços com os algoritmos de ordenação possui o sistema operacional Windows 7, CPU Intel Core 2 Duo T6600 de 2.20Ghz por núcleo e 4Gb de memória. Dois dispositivos móveis foram utilizados nos experimentos, sendo que o aplicativo gerado roda sem adaptações nos dois modelos:

- Samsung Galaxy® S5 com Sistema Operacional Android 6.0.1 (Marshmallow), bateria de 2800mAh 3,85V 10,78Wh, 2 processadores, Octa-Core, 2,1Ghz Quad-Core ARM Cortex-A15 e 1,5Ghz Quad-Core ARM Cortex-A7 e 2Gb de memória LPDDR3
- Samsung Galaxy S com Sistema Operacional Android 4.0.4 (Ice Cream Sandwich), bateria de 1500mAh 3,8V 5,70Wh, um processador, mono-core, 1Ghz ARM Cortex-A5 e 768 MB de memória LPDDR2.

3.2 Métodos

A Figura 5 apresenta um diagrama de atividades que ilustra a execução do aplicativo Android implementado para realizar os experimentos combinando as diferentes estratégias de comunicação, diferentes algoritmos de ordenação com diferentes tamanhos e tipos de entrada. No início, uma configuração de execução é escolhida. Uma configuração de execução é constituída por:

- um algoritmo de ordenação a ser executado;
- o tamanho do vetor de entrada (1.000, 10.000 ou 100.000 elementos);
- o tipo de entrada (Int, Float ou Object);
- se será considerado o melhor ou o pior caso;

- e se a execução será local ou remota (REST, SOAP, Socket ou gRPC).

Em seguida, coleta-se o nível de bateria atual do dispositivo. Todos os experimentos iniciam-se quando a bateria do dispositivo móvel está em sua capacidade máxima, portanto a primeira coleta de nível de bateria irá indicar que o nível está em 100%.

Figura 5 – Diagrama de atividade do aplicativo automatizado utilizado nos experimentos

No momento seguinte, o vetor a ser ordenado é criado de acordo com a configuração escolhida (tamanho, tipo do dado e melhor ou pior caso). Finalmente, o algoritmo é executado para ordenar o vetor criado no passo anterior e o nível de bateria é coletado novamente.

O processo de criar executar o algoritmo se repete até que a bateria tenha sido consumida em 1%. Se neste momento a bateria estiver com capacidade acima de 10%, uma nova configuração de execução é selecionada e o processo se repete até que a bateria tenha carga inferior a 10%. Neste caso, o dispositivo móvel é recarregado para que uma nova série de experimentos seja executada.

A interface BatteryManager do Android só retorna valores inteiros do nível de bateria do dispositivo. Desta forma, não era viável executar um algoritmo um número fixo de vezes e então coletar o consumo da bateria, pois não seria possível saber se o consumo foi de 0.6%, por exemplo, ou de 1%. Portanto, decidiu-se executar várias vezes o mesmo algoritmo até que o consumo fosse 1%. De acordo com as propriedades do dispositivo utilizado, 1% de bateria equivale a 388,1J para o modelo Galaxy S5 e 205,2J para o Galaxy S, visto que 1KW/h equivale a 3,6MJ. O consumo de energia é inversamente proporcional ao número de ordenações possíveis, ou seja, quanto mais ordenações possíveis em 1% de consumo de bateria, menor é seu consumo energético, portanto, melhor sua eficiência energética.

3.3 Ameaças à validade

A pesquisa experimental proposta possui algumas ameaças à validade que foram mitigadas pela repetição da execução de cada experimento e um intervalo de confiança de 95% foi considerado:

- **A energia consumida ao executar a mesma configuração pode variar na execução local:** as execuções locais dos experimentos podem ser influenciadas pelo kernel do sistema operacional ou outras rotinas em segundo plano;
- **A energia consumida ao executar a mesma configuração pode variar na execução remota:** durante as execuções remotas dos experimentos, a latência da rede pode oscilar;

- **Não linearidade das baterias de lítio:** Cada configuração de execução do experimento não foi executada aleatoriamente. Devido à não linearidade das baterias de lítio (HARPER; BURESS, 2008), o resultado da execução de uma configuração com nível de bateria em 100% pode ser diferente com o nível da bateria em 30%.
- **Quantidade de dispositivos utilizados:** o experimento foi realizado somente com dois celulares.

4 Resultados e discussões

Neste capítulo, os resultados dos experimentos realizados nesta pesquisa são apresentados e discutidos. Todas as análises verificam a relação entre a quantidade de dados recebida pelos algoritmos, o tipo do dado e a influência da complexidade assintótica dos algoritmos de ordenação utilizados.

Os resultados são apresentados por dispositivo móvel específico, Galaxy S5 e Galaxy S e analisa-se o comportamento do consumo de energia com os diferentes tipos de dados, tamanhos de entrada e complexidades assintóticas das execuções locais, remotas e por estratégia de comunicação possíveis em *computation offloading* (REST, SOAP, Socket e gRPC). Também analisa-se as circunstâncias em que as execuções locais são mais econômicas do que as execuções remotas, que as execuções remotas são mais econômicas do que as execuções locais e por fim as estratégias de comunicação que menos consumiram energia, neste caso, como o objetivo é comparar o consumo de energia ao utilizar estratégias como REST, SOAP, Socket e gRPC em diferentes circunstâncias, os valores do consumo de energia das execuções locais foram removidas e considerou-se apenas execuções remotas. Assim, para melhor organização, as seguintes questões de pesquisas foram elaboradas:

- QP1: Qual a influência do tipo de dado no consumo de energia?
- QP2: Qual a influência da quantidade de dados processados no consumo de energia?
- QP3: Qual a influência da complexidade assintótica do algoritmo no consumo de energia?
- QP4: Em quais situações as execuções locais consomem menos energia do que as execuções remotas?
- QP5: Em quais situações as execuções remotas consomem menos energia do que as execuções locais?
- QP6: Em quais situações as execuções de comunicação usadas em *computation offloading* consumiram menos energia?

Após a análise às questões de pesquisa acima, tanto com relação ao Galaxy S5 quanto ao Galaxy S, foi realizada uma análise comparando os resultados obtidos para verificar os comportamentos comuns resultantes das execuções locais, remotas e de cada estratégia de comunicação.

Por fim, as considerações finais deste capítulo são apresentadas.

Neste capítulo, somente os resultados pertinentes a cada discussão específica são apresentados. A tabulação de todos os resultados obtidos na execução do experimento com o celular Galaxy S5 e Galaxy S podem ser vistos, respectivamente, no Apêndice A e Apêndice B desta dissertação.

4.1 Galaxy S5

Os resultados obtidos com a execução dos experimentos no Galaxy S5 são apresentados e discutidos a seguir, organizados por questão de pesquisa.

Duas observações sobre os dados apresentados são necessárias. Em primeiro lugar, não foi possível concluir a execução local de algumas configurações: i) execução do Selection Sort, no pior ou melhor caso, ambos com complexidade assintótica $\mathcal{O}(n^2)$, sobre um vetor de 100.000 elementos do tipo Object; e ii) execução do Bubble Sort, no pior caso, portanto com complexidade assintótica $\mathcal{O}(n^2)$, também sobre um vetor de 100.000 elementos do tipo Object. Nessas duas situações, a memória disponível no dispositivo móvel não foi suficiente para executar a operação, o que ocasionou uma falha na aplicação. A segunda observação é sobre configurações em que não foi possível concluir uma única execução considerando o limite de 1% de bateria. Nesses casos, atribuiu-se o valor necessário para consumir 1% de bateria. Considerando o Galaxy S5, a energia presente em 1% de bateria é de 388,1000 Joules.

A Tabela 4 mostra os resultados da execução dos algoritmos de ordenação escolhidos para este estudo, considerando diferentes tamanhos de entrada, tipos de dados e situação do vetor a ser ordenado (melhor ou pior caso).

Com relação a Tabela 4, as 31 primeiras linhas apresentam as circunstâncias em que a execução local é mais econômica do que a execução remota e a partir da 32ª linha as circunstâncias em que a execução remota é mais econômica que a local são apresentadas. As Colunas 1 a 5 mostram a configuração de execução, enquanto a Coluna 6 mostra o consumo de energia quando o algoritmo é executado localmente e a Coluna 6 mostra o consumo de energia da execução remota mais econômica. A Coluna 7, por fim, mostra qual tipo de execução foi mais econômica para cada configuração: a execução remota ou local. Note que nesta tabela não estão apresentados os resultados para cada tecnologia ou arquitetura de comunicação, pois o objetivo é apenas mostrar em que circunstâncias a

execução remota pode ser mais econômica do que a execução local, independentemente de estratégia de comunicação, e como é o consumo de energia local considerando todas as configurações.

Tabela 4 – Comparação entre o gasto de energia das execuções locais e as execuções remotas (*offloading*) que menos consumiram energia no Galaxy S5

Algoritmo	Tipo dado	Tamanho entrada	Caso	Complexidade	Consumo energia execução local (Joule)	Consumo energia execução remota (Joule)	Menor consumo energia
Bubble Sort	Int	1.000	Melhor Caso	$O(n)$	0,0066	0,0971	Local
Bubble Sort	Float	1.000	Melhor Caso	$O(n)$	0,0788	0,0909	Local
Bubble Sort	Object	1.000	Melhor Caso	$O(n)$	0,1881	0,4674	Local
Bubble Sort	Int	10.000	Melhor Caso	$O(n)$	0,0562	0,3842	Local
Bubble Sort	Float	10.000	Melhor Caso	$O(n)$	0,6806	0,7419	Local
Bubble Sort	Object	10.000	Melhor Caso	$O(n)$	2,0480	6,1701	Local
Bubble Sort	Int	100.000	Melhor Caso	$O(n)$	0,5633	5,1541	Local
Bubble Sort	Float	100.000	Melhor Caso	$O(n)$	6,3832	6,6799	Local
Bubble Sort	Object	100.000	Melhor Caso	$O(n)$	20,1610	70,5636	Local
Heap Sort	Int	1.000	Melhor Caso	$O(n \log n)$	0,0120	0,1105	Local
Heap Sort	Int	1.000	Pior Caso	$O(n \log n)$	0,0084	0,0979	Local
Heap Sort	Float	1.000	Melhor Caso	$O(n \log n)$	0,0846	0,1164	Local
Heap Sort	Float	1.000	Pior Caso	$O(n \log n)$	0,0705	0,1379	Local
Heap Sort	Object	1.000	Melhor Caso	$O(n \log n)$	0,2069	0,4666	Local
Heap Sort	Object	1.000	Pior Caso	$O(n \log n)$	0,2785	0,5862	Local
Heap Sort	Int	10.000	Melhor Caso	$O(n \log n)$	0,0989	0,5654	Local
Heap Sort	Int	10.000	Pior Caso	$O(n \log n)$	0,0852	0,5927	Local
Heap Sort	Float	10.000	Pior Caso	$O(n \log n)$	0,7362	0,7587	Local
Heap Sort	Object	10.000	Melhor Caso	$O(n \log n)$	2,3839	3,7863	Local
Heap Sort	Object	10.000	Pior Caso	$O(n \log n)$	2,6082	6,7561	Local
Heap Sort	Int	100.000	Melhor Caso	$O(n \log n)$	1,0054	4,9058	Local
Heap Sort	Int	100.000	Pior Caso	$O(n \log n)$	0,9592	3,9846	Local
Heap Sort	Object	100.000	Melhor Caso	$O(n \log n)$	22,3046	44,8823	Local
Heap Sort	Object	100.000	Pior Caso	$O(n \log n)$	23,6646	57,9254	Local
Bubble Sort	Int	1.000	Pior Caso	$O(n^2)$	0,0273	0,1032	Local
Bubble Sort	Float	1.000	Pior Caso	$O(n^2)$	0,1035	0,1103	Local
Selection Sort	Int	1.000	Melhor Caso	$O(n^2)$	0,0286	0,1073	Local
Selection Sort	Int	1.000	Pior Caso	$O(n^2)$	0,0144	0,0808	Local
Selection Sort	Float	1.000	Melhor Caso	$O(n^2)$	0,0908	0,1044	Local
Selection Sort	Float	1.000	Pior Caso	$O(n^2)$	0,0859	0,1101	Local
Selection Sort	Int	10.000	Pior Caso	$O(n^2)$	0,8761	1,1707	Local
Heap Sort	Float	10.000	Melhor Caso	$O(n \log n)$	0,7628	0,6607	Offloading
Heap Sort	Float	100.000	Melhor Caso	$O(n \log n)$	6,9552	5,5922	Offloading
Heap Sort	Float	100.000	Pior Caso	$O(n \log n)$	8,4738	6,1408	Offloading
Bubble Sort	Object	1.000	Pior Caso	$O(n^2)$	1,1236	0,6971	Offloading
Bubble Sort	Int	10.000	Pior Caso	$O(n^2)$	2,3986	1,1022	Offloading
Bubble Sort	Float	10.000	Pior Caso	$O(n^2)$	4,8271	1,7052	Offloading
Bubble Sort	Object	10.000	Pior Caso	$O(n^2)$	84,3696	9,7146	Offloading
Bubble Sort	Int	100.000	Pior Caso	$O(n^2)$	242,5625	36,2710	Offloading
Bubble Sort	Float	100.000	Pior Caso	$O(n^2)$	388,1000	37,6796	Offloading
Bubble Sort	Object	100.000	Pior Caso	$O(n^2)$	-	388,1000	Offloading
Selection Sort	Object	1.000	Melhor Caso	$O(n^2)$	0,6567	0,5887	Offloading
Selection Sort	Object	1.000	Pior Caso	$O(n^2)$	0,5905	0,5746	Offloading
Selection Sort	Int	10.000	Melhor Caso	$O(n^2)$	0,8400	0,7021	Offloading
Selection Sort	Float	10.000	Melhor Caso	$O(n^2)$	1,9137	0,8871	Offloading
Selection Sort	Object	10.000	Melhor Caso	$O(n^2)$	47,3293	5,0370	Offloading
Selection Sort	Float	10.000	Pior Caso	$O(n^2)$	2,3450	0,9733	Offloading
Selection Sort	Object	10.000	Pior Caso	$O(n^2)$	44,7808	5,7710	Offloading
Selection Sort	Int	100.000	Melhor Caso	$O(n^2)$	66,9138	11,4822	Offloading
Selection Sort	Float	100.000	Melhor Caso	$O(n^2)$	121,2813	16,0041	Offloading
Selection Sort	Object	100.000	Melhor Caso	$O(n^2)$	-	215,6111	Offloading
Selection Sort	Int	100.000	Pior Caso	$O(n^2)$	88,2045	23,3795	Offloading
Selection Sort	Float	100.000	Pior Caso	$O(n^2)$	194,0500	22,9645	Offloading
Selection Sort	Object	100.000	Pior Caso	$O(n^2)$	-	250,3871	Offloading

Fonte: Carolina Chamas, 2017

4.1.1 QP1: Qual a influência do tipo de dado no consumo de energia?

Considerando as circunstâncias em que a execução local é mais econômica energeticamente do que a execução remota, apresentadas na Tabela 4, pode-se perceber uma evidente influência do tipo de dado no consumo de energia. Entre os tipos primitivos, Int e Float, o inteiro consumiu menos energia. Em contraste, o tipo Object consumiu muito mais energia que ambos.

Para melhor visualização, a Figura 6 mostra a soma dos valores de todas as execuções locais por tipo de dado, na qual se pode observar que o tipo de dado Object consome muito mais energia que Int e Float. Note que o tipo Object consumiu 84,78% do total geral da energia gasta de todas as execuções, enquanto que Float consumiu 29% e Int 16%.

Figura 6 – Consumo de energia dos tipos de dados de entrada das execuções locais com o celular Galaxy S5

Fonte: Carolina Chamas, 2017

Analisando os resultados em que a execução remota é mais econômica que a execução local, apresentadas na Tabela 4, o mesmo comportamento das execuções locais pode ser

visto nas execuções remotas. Além disso, observando a Figura 7, que exibe a soma do consumo de energia das execuções remotas e das execuções locais, as execuções remotas consumiram menos energia que as execuções locais para a maioria todos os tipos de dados. Isso mostra que o *computation offloading* tende a reduzir o consumo de energia, independente do tipo de dado.

Figura 7 – Comparação do consumo de energia das execuções locais e as execuções remotas (*offloading*) mais econômica no Galaxy S5 com relação aos tipos dos dados

Fonte: Carolina Chamas, 2017

Verificando a influência dos tipos de dados no consumo de energia nas diferentes estratégias de comunicação usadas em *computation offloading*, na Figura 8, a qual exibe a soma do consumo de energia dos diferentes tipos de dados, REST é o que menos consome energia com o tipo de dado Object, enquanto que Socket tende a ser o que menos consome energia com os tipos primitivos Int e Float. É importante notar que Object é o tipo de dado que mais consumiu energia em todas as estratégias de comunicação. Apesar de entre os tipos primitivos a diferença do consumo de energia ser menor, percebe-se que Float consome um pouco mais de energia que Int.

Figura 8 – Consumo de energia das execuções remotas (*offloading*) no Galaxy S5 com relação aos tipos dos dados

Fonte: Carolina Chamas, 2017

4.1.2 QP2: Qual a influência da quantidade de dados processados no consumo de energia?

Considerando as circunstâncias em que a execução local é mais econômica energeticamente do que a execução remota, apresentadas na Tabela 4, o tamanho de entrada do algoritmo exerce um papel fundamental neste contexto. A quantidade de energia consumida aumenta consideravelmente de 1.000 elementos para 10.000 e, principalmente, em seguida, para 100.000.

A Figura 9 exibe o total de energia consumida de todas as execuções locais. É possível observar que a entrada com tamanho de 100.000 elementos consome muito mais energia que os outros tamanhos de entrada (92%), enquanto que a entrada com 10.000 elementos consome 8% de energia e a energia consumida com as execuções com tamanho de 1.000 é quase insignificante, chegando próximo de 0% do total.

Figura 9 – Consumo de energia dos tamanhos de entrada das execuções locais com o celular Galaxy S5

Fonte: Carolina Chamas, 2017

Verificando os resultados em que a execução remota é mais econômica que a execução local, apresentadas na Tabela 4, o mesmo comportamento das execuções locais pode ser observado, ou seja, conforme o tamanho de entrada aumenta, o consumo de energia também aumenta. A Figura 10 exibe a soma geral do gasto de energia dos diferentes tamanhos de entrada das execuções locais, é interessante observar que comparado às execuções locais, as remotas, com todos tamanhos de entrada, tendem a consumir menos energia.

Figura 10 – Comparação do consumo de energia das execuções locais e as execuções remotas (*offloading*) mais econômica no Galaxy S5 com relação aos tamanhos de entrada do algoritmo

Fonte: Carolina Chamas, 2017

Analisando a influência dos tamanhos de entrada no consumo de energia nas diferentes estratégias de comunicação usadas em *computation offloading*, na Figura 11, a qual exibe a soma do consumo de energia dos diferentes tamanhos de entrada, REST tende a consumir menos energia com entrada pequena de 1.000 e 100.000 elementos e Socket com 10.000 e também com 100.000 elementos. O tamanho de 100.000 elementos foi o que mais consumiu energia, seguido por 10.000 e então 1.000.

Figura 11 – Consumo de energia das execuções remotas (*offloading*) no Galaxy S5 com relação aos tamanhos de entrada

Fonte: Carolina Chamas, 2017

4.1.3 QP3: Qual a influência da complexidade assintótica do algoritmo no consumo de energia?

Considerando as circunstâncias em que a execução local é mais econômica energeticamente do que a execução remota, apresentadas na Tabela 4, a complexidade assintótica dos algoritmos influencia no consumo de energia, sendo que o algoritmo com complexidade

$\mathcal{O}(n)$ consumiu menos energia do que a $\mathcal{O}(n \log n)$, que por sua vez consumiu menos energia do que os algoritmos de complexidade $\mathcal{O}(n^2)$.

Como melhor visualizado na Figura 12, 96% do total de energia gasta foi consumida por algoritmos mais complexos $\mathcal{O}(n^2)$, seguido por $\mathcal{O}(n \log n)$ consumindo 3% e, por fim, a complexidade $\mathcal{O}(n)$ consumindo 1%.

Figura 12 – Consumo de energia das complexidades dos algoritmos das execuções locais com o celular Galaxy S5

Fonte: Carolina Chamas, 2017

O mesmo comportamento pode ser observado nas execuções remotas, conforme Tabela 4, à medida que a complexidade assintótica do algoritmo aumenta, o consumo de energia também aumenta.

Comparando os valores da soma do consumo de energia das execuções remotas com os valores da soma do consumo de energia das execuções locais, apresentados na Figura 13, pode-se notar que o consumo de energia para as complexidades $\mathcal{O}(n)$ e $\mathcal{O}(n \log n)$ são menores nas execuções locais. Esses valores mostram uma tendência das complexidades $\mathcal{O}(n)$ e $\mathcal{O}(n \log n)$ consumirem menos energia quando executadas localmente. Para a complexidade $\mathcal{O}(n^2)$ essa tendência muda, mostrando que pode existir uma redução do consumo de energia executando remotamente algoritmos com essa complexidade.

Figura 13 – Comparação do consumo de energia das execuções locais e as execuções remotas (*offloading*) mais econômica no Galaxy S5 com relação as complexidades assintóticas

Fonte: Carolina Chamas, 2017

Verificando a influência das complexidades assintóticas no consumo de energia nas diferentes estratégias de comunicação usadas em *computation offloading*, na Figura 14, a qual exibe a soma do consumo de energia das diferentes complexidades assintóticas, REST tende a consumir menos energia com a complexidade $O(n^2)$ e $O(n)$ e Socket com a complexidade $O(n \log n)$. O consumo de energia, em quase todas as estratégias de comunicação, é maior nas execuções $O(n^2)$, seguido por $O(n \log n)$ e a que menos consome sendo $O(n)$. Existe uma exceção com a execução com a tecnologia Socket, a qual consumiu menos energia em algumas situações com a complexidade $O(n \log n)$ que $O(n)$.

Figura 14 – Consumo de energia das execuções remotas (*offloading*) no Galaxy S5 com relação as complexidades assintóticas

Fonte: Carolina Chamas, 2017

Figura 15 – Gráficos da comparação do consumo de energia das execuções locais e as execuções remotas (*offloading*) do Galaxy S5

4.1.4 QP4: Em quais situações as execuções locais consomem menos energia do que as execuções remotas?

De maneira geral, é possível identificar, pela análise da Tabela 4, em quais situações as execuções locais consomem menos energia do que as execuções remotas, que podem ser melhores visualizados na Figura 15:

- Algoritmos com complexidade assintótica linear $\mathcal{O}(n)$, independente da quantidade e tipo de dado;
- Majoritariamente algoritmos com complexidade assintótica $\mathcal{O}(n \log n)$;
- Algoritmos com complexidade assintótica $\mathcal{O}(n^2)$, desde que a quantidade de dados seja pequena (1.000 elementos) e associadas a tipos de dados primitivos, Int e Float.

4.1.5 QP5: Em quais situações as execuções remotas consomem menos energia do que as execuções locais?

Analisando mais detalhadamente os dados apresentados na Tabela 4 e melhor visualizados na Figura 15, é possível perceber que as execuções remotas são mais eficazes no que se refere ao consumo de energia nas seguintes circunstâncias:

- Majoritariamente para complexidade assintótica quadrática, associada principalmente a tamanho de vetores a partir de 10.000 elementos;
- Tamanho de entrada pequeno (1.000) associado a complexidade assintótica quadrática e tipo Object.

4.1.6 QP6: Em quais situações as execuções de comunicação usadas em *computation offloading* consumiram menos energia?

Os dados das Tabelas 5, 6, 8 e 7, respectivamente, apresentam as situações em que o REST, Socket, SOAP e gRPC foram as alternativas de comunicação com a rede mais econômicas.

No geral, a alternativa mais eficiente energeticamente foi o REST, que, conforme pode ser visualizado na Figura 16, consumiu menos energia em 32 vezes, ou seja, 59% dos casos. O Socket foi a segunda alternativa mais econômica, pois consumiu menos energia em 15 situações distintas (aproximadamente 28% dos casos). Já o SOAP e o gRPC foram

as estratégias de comunicação que menos vezes foram a alternativa mais econômica. O SOAP foi a opção mais econômica 5 vezes (aproximadamente 9% dos casos), enquanto o gRPC apenas 2 vezes (aproximadamente 4% dos casos).

Figura 16 – Consumo de energia das estratégias de comunicação utilizadas em *computation offloading* do celular Galaxy S5

Fonte: Carolina Chamas, 2017

A análise dos casos em que o REST consumiu menos energia, conforme apresentados na Tabela 5 e como pode ser observado na Figura 17, possibilitou encontrar situações de referência em que esta forma de comunicação pode ser mais econômica.

A seguir são apresentadas as configurações mais comuns para as quais o REST foi mais econômico:

- Com o tipo de dado Object, independentemente do tamanho de entrada e da complexidade assintótica do algoritmo;
- Com entradas pequenas, com 1.000 elementos, independente do tipo de dado e complexidade;

Tabela 5 – Execuções em que o REST foi a alternativa mais econômica - Galaxy S5

Algoritmos	Tipo dado	Tamanho entrada	Caso	Complexidade	Consumo energia execuções com REST
Bubble Sort	Int	1.000	Melhor Caso	$\mathcal{O}(n)$	0,0971
Bubble Sort	Int	10.000	Melhor Caso	$\mathcal{O}(n)$	0,3842
Bubble Sort	Object	1.000	Melhor Caso	$\mathcal{O}(n)$	0,4674
Bubble Sort	Object	10.000	Melhor Caso	$\mathcal{O}(n)$	6,1701
Bubble Sort	Object	100.000	Melhor Caso	$\mathcal{O}(n)$	70,5636
Heap Sort	Int	1.000	Pior Caso	$\mathcal{O}(n \log n)$	0,0979
Heap Sort	Int	1.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,1105
Heap Sort	Int	10.000	Pior Caso	$\mathcal{O}(n \log n)$	0,5927
Heap Sort	Int	10.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,5654
Heap Sort	Int	100.000	Melhor Caso	$\mathcal{O}(n \log n)$	4,9058
Heap Sort	Float	1.000	Pior Caso	$\mathcal{O}(n \log n)$	0,1379
Heap Sort	Float	1.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,1164
Heap Sort	Object	1.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,4666
Heap Sort	Object	1.000	Pior Caso	$\mathcal{O}(n \log n)$	0,5862
Heap Sort	Object	10.000	Melhor Caso	$\mathcal{O}(n \log n)$	3,7863
Heap Sort	Object	100.000	Melhor Caso	$\mathcal{O}(n \log n)$	44,8823
Bubble Sort	Float	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,1103
Bubble Sort	Float	10.000	Pior Caso	$\mathcal{O}(n^2)$	1,7052
Bubble Sort	Object	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,6971
Bubble Sort	Object	10.000	Pior Caso	$\mathcal{O}(n^2)$	9,7146
Bubble Sort	Object	100.000	Pior Caso	$\mathcal{O}(n^2)$	388,1000
Selection Sort	Int	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,0808
Selection Sort	Int	1.000	Melhor Caso	$\mathcal{O}(n^2)$	0,1073
Selection Sort	Float	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,1101
Selection Sort	Float	10.000	Pior Caso	$\mathcal{O}(n^2)$	0,9733
Selection Sort	Float	100.000	Melhor Caso	$\mathcal{O}(n^2)$	16,0041
Selection Sort	Object	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,5746
Selection Sort	Object	1.000	Melhor Caso	$\mathcal{O}(n^2)$	0,5887
Selection Sort	Object	10.000	Pior Caso	$\mathcal{O}(n^2)$	5,7710
Selection Sort	Object	10.000	Melhor Caso	$\mathcal{O}(n^2)$	5,0370
Selection Sort	Object	100.000	Melhor Caso	$\mathcal{O}(n^2)$	215,6111
Selection Sort	Object	100.000	Pior Caso	$\mathcal{O}(n^2)$	250,3871

Fonte: Carolina Chamas, 2017

O Socket também foi a alternativa mais econômica em um número significativo de vezes, cerca de 28% de casos. Portanto, também foi possível, com base nos dados apresentados na Tabela 6 e no comportamento observado na Figura 17, extrair as seguintes tendências no que se refere aos casos em que esta forma de comunicação consome menos energia:

- Execuções com grande volume de dados, com 100.000 elementos, desde que associadas a tipos primitivos Int e Float;
- Execuções com 10.000 elementos podem consumir menos energia se associadas a tipos de dados primitivos, mas não é um padrão definitivo;

Figura 17 – Gráficos da comparação do consumo de energia das execuções remotas (*offloading*) do Galaxy S5

Fonte: Carolina Chamas, 2017

Tabela 6 – Execuções em que o Socket foi a alternativa mais econômica - Galaxy S5

Algoritmos	Tipo dado	Tamanho entrada	Caso	Complexidade	Consumo energia execuções com Socket
Bubble Sort	Int	100.000	Melhor Caso	$\mathcal{O}(n)$	5,1541
Bubble Sort	Float	10.000	Melhor Caso	$\mathcal{O}(n)$	0,7419
Bubble Sort	Float	100.000	Melhor Caso	$\mathcal{O}(n)$	6,6799
Heap Sort	Int	100.000	Pior Caso	$\mathcal{O}(n \log n)$	3,9846
Heap Sort	Float	10.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,6607
Heap Sort	Float	100.000	Melhor Caso	$\mathcal{O}(n \log n)$	5,5922
Heap Sort	Float	100.000	Pior Caso	$\mathcal{O}(n \log n)$	6,1408
Heap Sort	Object	10.000	Pior Caso	$\mathcal{O}(n \log n)$	6,7561
Heap Sort	Object	100.000	Pior Caso	$\mathcal{O}(n \log n)$	57,9254
Bubble Sort	Int	10.000	Pior Caso	$\mathcal{O}(n^2)$	1,1022
Bubble Sort	Float	100.000	Pior Caso	$\mathcal{O}(n^2)$	37,6796
Selection Sort	Int	10.000	Melhor Caso	$\mathcal{O}(n^2)$	0,7021
Selection Sort	Int	100.000	Melhor Caso	$\mathcal{O}(n^2)$	11,4822
Selection Sort	Float	10.000	Melhor Caso	$\mathcal{O}(n^2)$	0,8871
Selection Sort	Float	100.000	Pior Caso	$\mathcal{O}(n^2)$	22,9645

Fonte: Carolina Chamas, 2017

As circunstâncias em que o gRPC e o SOAP foram as alternativas mais econômicas, conforme apresentadas na Tabela 7 e 8, respectivamente, não permitem uma análise mais detalhada sobre situações em que este tipo de comunicação com a rede tende a ser mais econômica, principalmente pelo número limitado de vezes em que essas alternativas foram mais econômicas do que as outras.

Tabela 7 – Execuções em que o gRPC foi a alternativa mais econômica - Galaxy S5

Algoritmos	Tipo dado	Tamanho entrada	Caso	Complexidade	Consumo energia execuções com gRPC
Bubble Sort	Float	1.000	Melhor Caso	$\mathcal{O}(n)$	0,0909
Bubble Sort	Int	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,1032

Fonte: Carolina Chamas, 2017

Tabela 8 – Execuções em que o SOAP foi a alternativa mais econômica - Galaxy S5

Algoritmos	Tipo dado	Tamanho entrada	Caso	Complexidade	Consumo energia execuções com SOAP
Heap Sort	Float	10.000	Pior Caso	$\mathcal{O}(n \log n)$	0,7587
Bubble Sort	Int	100.000	Pior Caso	$\mathcal{O}(n^2)$	36,2710
Selection Sort	Int	10.000	Pior Caso	$\mathcal{O}(n^2)$	1,1707
Selection Sort	Int	100.000	Pior Caso	$\mathcal{O}(n^2)$	23,3795
Selection Sort	Float	1.000	Melhor Caso	$\mathcal{O}(n^2)$	0,1044

Fonte: Carolina Chamas, 2017

4.2 *Galaxy S*

Os resultados obtidos com a execução dos experimentos no Galaxy S são apresentados e discutidos a seguir, organizados por questão de pesquisa.

Uma observação sobre os dados apresentados é necessária. Não foi possível concluir a execução de algumas configurações: i) independente da complexidade assintótica do algoritmo, sobre um vetor de 100.000 elementos do tipo Object; e ii) com a estratégia de comunicação gRPC. Nessas duas situações, a memória disponível no dispositivo móvel não foi suficiente para executar a operação, o que ocasionou uma falha na aplicação.

A Tabela 4 mostra os resultados da execução dos algoritmos de ordenação escolhidos para este estudo, considerando diferentes tamanhos de entrada, tipos de dados e situação do vetor a ser ordenado (melhor ou pior caso).

Com relação a Tabela 9, as 20 primeiras linhas apresentam as circunstâncias em que a execução local é mais econômica do que a execução remota e a partir da 21^a linha as circunstâncias em que a execução remota é mais econômica que a local são apresentadas. As Colunas 1 a 5 mostram a configuração de execução, enquanto a Coluna 6 mostra o consumo de energia quando o algoritmo é executado localmente e a Coluna 6 mostra o consumo de energia da execução remota mais econômica. A Coluna 7, por fim, mostra qual tipo de execução foi mais econômica para cada configuração: a execução remota ou local. Note que nesta tabela não estão apresentados os resultados para cada tecnologia ou arquitetura de comunicação, pois o objetivo é apenas mostrar em que circunstâncias a execução remota pode ser mais econômica do que a execução local, independentemente de estratégia de comunicação, e como é o consumo de energia local considerando todas as configurações.

4.2.1 QP1: Qual a influência do tipo de dado no consumo de energia?

Considerando as circunstâncias em que a execução local é mais econômica do que a execução remota, apresentadas na Tabela 9, pode-se perceber uma evidente influência do tipo de dado no consumo de energia. Entre os tipos primitivos, Int e Float, o inteiro consumiu menos energia. Em contraste, o tipo Object consumiu muito mais energia que ambos.

Tabela 9 – Comparação entre o gasto de energia das execuções locais contra o gasto de energia da execução remota (*offloading*) que menos consumiram energia no Galaxy S

Algoritmos	Tipo dado	Tamanho entrada	Caso	Complexidade	Consumo energia execução local	Consumo energia execução remota	Menor consumo energia
Bubble Sort	Int	1.000	Melhor Caso	$\mathcal{O}(n)$	0,0638	0,1638	Local
Bubble Sort	Float	1.000	Melhor Caso	$\mathcal{O}(n)$	0,1025	0,1465	Local
Bubble Sort	Object	1.000	Melhor Caso	$\mathcal{O}(n)$	0,4111	1,6157	Local
Bubble Sort	Int	10.000	Melhor Caso	$\mathcal{O}(n)$	0,7714	1,0527	Local
Bubble Sort	Float	10.000	Melhor Caso	$\mathcal{O}(n)$	0,8073	1,1520	Local
Bubble Sort	Object	10.000	Melhor Caso	$\mathcal{O}(n)$	3,3405	8,6052	Local
Heap Sort	Int	1.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,1121	0,1350	Local
Heap Sort	Float	1.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,1172	0,1228	Local
Heap Sort	Object	1.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,4815	1,3737	Local
Heap Sort	Object	1.000	Pior Caso	$\mathcal{O}(n \log n)$	0,5507	1,1726	Local
Heap Sort	Int	10.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,9365	1,1503	Local
Heap Sort	Int	10.000	Pior Caso	$\mathcal{O}(n \log n)$	0,9883	1,3473	Local
Heap Sort	Float	10.000	Melhor Caso	$\mathcal{O}(n \log n)$	1,2732	1,3460	Local
Heap Sort	Float	10.000	Pior Caso	$\mathcal{O}(n \log n)$	1,5605	1,6172	Local
Heap Sort	Object	10.000	Melhor Caso	$\mathcal{O}(n \log n)$	4,3660	10,8000	Local
Heap Sort	Object	10.000	Pior Caso	$\mathcal{O}(n \log n)$	4,2201	6,1102	Local
Bubble Sort	Int	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,1387	0,1402	Local
Selection Sort	Int	1.000	Melhor Caso	$\mathcal{O}(n^2)$	0,1273	0,1326	Local
Selection Sort	Object	1.000	Melhor Caso	$\mathcal{O}(n^2)$	1,0065	1,2094	Local
Selection Sort	Object	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,9544	0,9771	Local
Bubble Sort	Int	100.000	Melhor Caso	$\mathcal{O}(n)$	6,4376	5,6551	Offloading
Bubble Sort	Float	100.000	Melhor Caso	$\mathcal{O}(n)$	7,2424	4,9361	Offloading
Heap Sort	Int	1.000	Pior Caso	$\mathcal{O}(n \log n)$	0,0902	0,0570	Offloading
Heap Sort	Float	1.000	Pior Caso	$\mathcal{O}(n \log n)$	0,1363	0,1094	Offloading
Heap Sort	Int	100.000	Melhor Caso	$\mathcal{O}(n \log n)$	8,3330	7,9948	Offloading
Heap Sort	Int	100.000	Pior Caso	$\mathcal{O}(n \log n)$	8,4619	7,6000	Offloading
Heap Sort	Float	100.000	Melhor Caso	$\mathcal{O}(n \log n)$	13,9909	8,8364	Offloading
Heap Sort	Float	100.000	Pior Caso	$\mathcal{O}(n \log n)$	16,4160	5,4158	Offloading
Bubble Sort	Float	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,2041	0,1407	Offloading
Bubble Sort	Object	1.000	Pior Caso	$\mathcal{O}(n^2)$	1,7298	1,3562	Offloading
Bubble Sort	Int	10.000	Pior Caso	$\mathcal{O}(n^2)$	7,6000	1,4730	Offloading
Bubble Sort	Float	10.000	Pior Caso	$\mathcal{O}(n^2)$	10,8715	1,3024	Offloading
Bubble Sort	Object	10.000	Pior Caso	$\mathcal{O}(n^2)$	46,9029	5,4776	Offloading
Bubble Sort	Int	100.000	Pior Caso	$\mathcal{O}(n^2)$	205,2000	19,1914	Offloading
Bubble Sort	Float	100.000	Pior Caso	$\mathcal{O}(n^2)$	205,2000	27,3600	Offloading
Selection Sort	Float	1.000	Melhor Caso	$\mathcal{O}(n^2)$	0,1120	0,1033	Offloading
Selection Sort	Int	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,1201	0,0744	Offloading
Selection Sort	Float	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,1859	0,1075	Offloading
Selection Sort	Int	10.000	Pior Caso	$\mathcal{O}(n^2)$	3,0857	1,2595	Offloading
Selection Sort	Float	10.000	Melhor Caso	$\mathcal{O}(n^2)$	2,0167	0,7628	Offloading
Selection Sort	Float	10.000	Pior Caso	$\mathcal{O}(n^2)$	4,0634	1,0138	Offloading
Selection Sort	Object	10.000	Melhor Caso	$\mathcal{O}(n^2)$	14,8696	9,5126	Offloading
Selection Sort	Object	10.000	Pior Caso	$\mathcal{O}(n^2)$	17,8435	9,9910	Offloading
Selection Sort	Int	10.000	Melhor Caso	$\mathcal{O}(n^2)$	2,8649	0,8135	Offloading
Selection Sort	Int	100.000	Melhor Caso	$\mathcal{O}(n^2)$	143,6400	8,7526	Offloading
Selection Sort	Int	100.000	Pior Caso	$\mathcal{O}(n^2)$	205,2000	10,8000	Offloading
Selection Sort	Float	100.000	Melhor Caso	$\mathcal{O}(n^2)$	111,9273	14,2500	Offloading
Selection Sort	Float	100.000	Pior Caso	$\mathcal{O}(n^2)$	205,2000	16,4160	Offloading

Fonte: Carolina Chamas, 2017

Para melhor visualização, a Figura 18 mostra a soma de todos os valores das execuções locais por tipo de dado, lembrando que as execuções com tamanho de 100.000 elementos e tipo de dado Object por foram possíveis de serem executadas. Assim, pode-se

observar que o tipo de dado Object consumiu 72% do total da energia consumida, muito mais energia que Float(16%) e Int(12%).

Figura 18 – Consumo de energia dos tipos de dados das execuções locais com o celular Galaxy S

Fonte: Carolina Chamas, 2017

Analisando os resultados em que a execução remota é mais econômica que a execução local, apresentadas na Tabela 9, o mesmo comportamento das execuções locais pode ser visto nas execuções remotas. Além disso, observando a Figura 19, que exibe a soma das execuções remotas e das execuções locais, as execuções remotas consumiram menos energia que a soma das execuções locais para todos os tipos de dados. Isso mostra que o *computation offloading* tende a reduzir o consumo de energia, independente do tipo de dado. O consumo de energia com Object é menor que Int e Float, pois as configurações com 100.000 elementos com esses tipos de dados foram executadas, também a soma do consumo de energia do tipo de dado Float foi um pouco menor que a soma do tipo de dado Int em decorrência de que algumas execuções com a tecnologia Socket consumiu mais energia para executar tipos inteiros do que float em algumas situações. Isso fica evidenciado na Figura 20.

Verificando a influência dos tipos de dados no consumo de energia nas diferentes estratégias de comunicação usadas em *computation offloading*, na Figura 20, a qual exibe a soma do consumo de energia dos diferentes tipos de dados, REST tende a consumir menos energia com o tipo de dado Object e Int, enquanto que Socket consome menos energia com o tipo Float. É importante dizer que existem execuções com o tipo de dado Object que não foram possíveis de serem realizadas no Galaxy S, devido a isso, como foi realizada a soma do consumo de energia, este tipo de dado apresenta um baixo consumo de energia. Float tende a consumir menos energia quando comparado ao tipo Int com todas as estratégias de comunicação, entretando, algumas execuções com a tecnologias Socket consumiu menos energia com Float que com o tipo Int, o que fez com que a Figura 20 mostrasse um consumo de energia menor para esta tecnologia.

Figura 19 – Comparação do consumo de energia das execuções locais e as execuções remotas (*offloading*) mais econômica no Galaxy S com relação aos tipos dos dados

Fonte: Carolina Chamas, 2017

Figura 20 – Consumo de energia das execuções remotas (*offloading*) no Galaxy S com relação aos tipos dos dados

Fonte: Carolina Chamas, 2017

4.2.2 QP2: Qual a influência da quantidade de dados processados no consumo de energia?

Considerando as circunstâncias em que a execução local é mais econômica energeticamente do que a execução remota, apresentadas na Tabela 9, o tamanho de entrada do algoritmo exerce um papel fundamental neste contexto. A quantidade de energia consumida aumenta consideravelmente de 1.000 elementos para 10.000 e, principalmente, em seguida, para 100.000.

A Figura 21 exibe o total de energia consumida de todas as execuções locais, lembrando que as execuções com tamanho de 100.000 elementos e tipo de dado Object por foram possíveis de serem executadas.

É possível observar que a entrada com tamanho de 100.000 elementos consome muito mais energia que os outros tamanhos de entrada (97%), enquanto que a entrada com 10.000 elementos consome 3% de energia e a energia consumida com as execuções com tamanho de 1.000 é quase insignificante, chegando próximo de 0% do total.

Figura 21 – Consumo de energia dos tamanhos de entrada das execuções locais com o celular Galaxy S

Fonte: Carolina Chamas, 2017

Verificando os resultados em que a execução remota é mais econômica que a execução local, apresentadas na Tabela 9, o mesmo comportamento das execuções locais pode ser observado, ou seja, conforme o tamanho de entrada aumenta, o consumo de energia também aumenta. A Figura 22 exibe a soma geral do gasto de energia dos diferentes tamanhos de entrada das execuções locais, é interessante observar que comparado às execuções locais, as remotas, com todos tamanhos de entrada, tendem a consumir menos energia.

Figura 22 – Comparação do consumo de energia das execuções locais e as execuções remotas (*offloading*) mais econômica no Galaxy S com relação aos tamanhos dos dados

Fonte: Carolina Chamas, 2017

Verificando a influência dos tamanhos de entrada no consumo de energia nas diferentes estratégias de comunicação usadas em *computation offloading*, na Figura 23, a qual exibe a soma do consumo de energia dos diferentes tamanhos de entrada, REST tende a consumir menos energia com qualquer tamanho de dado.

Figura 23 – Consumo de energia das execuções remotas (*offloading*) no Galaxy S com relação aos tamanhos de entrada

Fonte: Carolina Chamas, 2017

4.2.3 QP3: Qual a influência da complexidade assintótica do algoritmo no consumo de energia?

Considerando as circunstâncias em que a execução local é mais econômica energeticamente do que a execução remota, apresentadas na Tabela 9, a complexidade assintótica dos algoritmos influencia no consumo de energia, sendo que o algoritmo com complexidade $\mathcal{O}(n)$ consumiu menos energia do que a $\mathcal{O}(n \log n)$, que por sua vez consumiu menos energia do que os algoritmos de complexidade $\mathcal{O}(n^2)$.

Como melhor visualizado na Figura 24, 94% do total de energia consumida é consumida por algoritmos mais complexos $\mathcal{O}(n^2)$, seguido por $\mathcal{O}(n \log n)$ consumindo 5% e, por fim, a complexidade $\mathcal{O}(n)$ consumindo 1%.

O mesmo comportamento pode ser observado nas execuções remotas, conforme Tabela 9, à medida que a complexidade assintótica do algoritmo aumenta, o consumo de energia também aumenta.

Figura 24 – Consumo de energia das complexidades dos algoritmos das execuções locais com o celular Galaxy S

Fonte: Carolina Chamas, 2017

Figura 25 – Comparação do consumo de energia das execuções locais e as execuções remotas (*offloading*) mais econômica no Galaxy S com relação as complexidades assintóticas

Fonte: Carolina Chamas, 2017

Comparando os valores da soma do consumo de energia das execuções remotas com os valores da soma do consumo de energia das execuções locais, conforme apresentados na Figura 25, pode-se notar que o consumo de energia para as complexidades $O(n)$ e $O(n \log n)$ são menores nas execuções locais. Esses valores mostram uma tendência das complexidades $O(n)$ e $O(n \log n)$ consumirem menos energia quando executadas localmente. Para a complexidade $O(n^2)$, essa tendência muda, mostrando que pode existir uma redução do consumo de energia executando remotamente algoritmos com esse nível de complexidade.

Verificando a influência das complexidades assintóticas no consumo de energia nas diferentes estratégias de comunicação usadas em *computation offloading*, na Figura 26, a qual exibe a soma do consumo de energia das diferentes complexidades assintóticas, REST tende a consumir menos energia com a complexidade $\mathcal{O}(n^2)$, enquanto que Socket tende a consumir menos energia com a complexidade $\mathcal{O}(n \log n)$ e $\mathcal{O}(n)$. O consumo de energia, em todas as estratégias de comunicação, é maior nas execuções $\mathcal{O}(n^2)$, seguido por $\mathcal{O}(n \log n)$ e a que menos consome sendo $\mathcal{O}(n)$.

4.2.4 QP4: Em quais situações as execuções locais consomem menos energia do que as execuções remotas?

De maneira geral, é possível, pela análise da Tabela 9, identificar em quais situações as execuções locais consomem menos energia do que as execuções remotas, que podem ser melhores visualizadas na Figura 27:

- entradas com 1.000 e 10.000 elementos, associadas a complexidade assintótica $\mathcal{O}(n)$ e $\mathcal{O}(n \log n)$ majoritariamente.

Figura 26 – Consumo de energia das execuções remotas (*offloading*) no Galaxy S com relação as complexidades assintóticas

Fonte: Carolina Chamas, 2017

4.2.5 QP5: Em quais situações as execuções remotas consomem menos energia do que as execuções locais?

Analisando mais detalhadamente os dados apresentados na Tabela 9 e que podem ser visualizados na Figura 27, é possível perceber que as execuções remotas são mais eficazes no que se refere ao consumo de energia nas seguintes circunstâncias:

- Todos os algoritmos, independentemente de sua complexidade assintótica, com entrada de 100.000 elementos;
- Majoritariamente os algoritmos de complexidade assintótica $\mathcal{O}(n^2)$.

4.2.6 QP6: Em quais situações as execuções de comunicação usadas em *computation offloading* consumiram menos energia?

Os dados das Tabelas 10, 11 e 12, respectivamente, apresentam as situações em que o REST, SOAP e Socket foram as alternativas de comunicação com a rede mais econômicas.

No geral, a alternativa mais eficiente energeticamente foi o REST, que, conforme pode ser visualizado na Figura 28, consumiu menos energia em 33 vezes, ou seja, 69% dos casos. O Socket foi a segunda alternativa mais econômica, pois consumiu menos energia em 13 situações distintas (aproximadamente 27% dos casos). Já o SOAP foi a estratégia de comunicação que menos vezes foi a alternativa mais econômica. O SOAP foi a opção mais econômica 2 vezes (aproximadamente 4% dos casos).

A análise dos casos em que o REST consumiu menos energia, conforme apresentados na Tabela 10 e como pode ser observado na Figura 29, possibilitou encontrar situações de referência em que esta forma de comunicação pode ser mais econômica. A seguir são apresentadas as configurações mais comuns para as quais o REST foi mais econômico:

- Com tipo de dado Object definitivamente foi o que consumiu menos energia, independentemente do tamanho de entrada e da complexidade assintótica do algoritmo;
- Execuções com 1.000 e 10.000 elementos, independente do tipo de dado e complexidade assintótica do algoritmo.
- Execuções com 100.000 elementos também pode consumir menos energia com tipos de dados primitivos como Int e Float.

Figura 27 – Gráficos da comparação do consumo de energia das execuções locais e as execuções remotas (*offloading*) do Galaxy S

Fonte: Carolina Chamas, 2017

Figura 28 – Consumo de energia das estratégias de comunicação utilizadas em *computation offloading* do celular Galaxy S

Fonte: Carolina Chamas, 2017

Figura 29 – Gráficos da comparação do consumo de energia das execuções remotas (*offloading*) do Galaxy S

Fonte: Carolina Chamas, 2017

Tabela 10 – Execuções remotas com REST - Galaxy S

Algoritmos	Tipo dado	Tamanho entrada	Caso	Complexidade	Consumo energia execução com REST
Bubble Sort	Int	10.000	Melhor Caso	$\mathcal{O}(n)$	1,0527
Bubble Sort	Float	1.000	Melhor Caso	$\mathcal{O}(n)$	0,1465
Bubble Sort	Object	1.000	Melhor Caso	$\mathcal{O}(n)$	1,6157
Bubble Sort	Object	10.000	Melhor Caso	$\mathcal{O}(n)$	8,6052
Heap Sort	Int	1.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,1350
Heap Sort	Int	1.000	Pior Caso	$\mathcal{O}(n \log n)$	0,0570
Heap Sort	Int	10.000	Melhor Caso	$\mathcal{O}(n \log n)$	1,1503
Heap Sort	Int	10.000	Pior Caso	$\mathcal{O}(n \log n)$	1,3473
Heap Sort	Float	1.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,1228
Heap Sort	Float	1.000	Pior Caso	$\mathcal{O}(n \log n)$	0,1094
Heap Sort	Float	10.000	Melhor Caso	$\mathcal{O}(n \log n)$	1,3460
Heap Sort	Object	1.000	Melhor Caso	$\mathcal{O}(n \log n)$	1,3737
Heap Sort	Object	1.000	Pior Caso	$\mathcal{O}(n \log n)$	1,1726
Heap Sort	Object	10.000	Melhor Caso	$\mathcal{O}(n \log n)$	10,8000
Heap Sort	Object	10.000	Pior Caso	$\mathcal{O}(n \log n)$	6,1102
Bubble Sort	Int	10.000	Pior Caso	$\mathcal{O}(n^2)$	1,4730
Bubble Sort	Int	100.000	Pior Caso	$\mathcal{O}(n^2)$	19,1914
Bubble Sort	Float	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,1407
Bubble Sort	Float	10.000	Pior Caso	$\mathcal{O}(n^2)$	1,3024
Bubble Sort	Object	1.000	Pior Caso	$\mathcal{O}(n^2)$	1,3562
Bubble Sort	Object	10.000	Pior Caso	$\mathcal{O}(n^2)$	5,4776
Selection Sort	Int	10.000	Pior Caso	$\mathcal{O}(n^2)$	1,2595
Selection Sort	Int	100.000	Melhor Caso	$\mathcal{O}(n^2)$	8,7526
Selection Sort	Int	100.000	Pior Caso	$\mathcal{O}(n^2)$	10,8000
Selection Sort	Float	1.000	Melhor Caso	$\mathcal{O}(n^2)$	0,1033
Selection Sort	Float	10.000	Melhor Caso	$\mathcal{O}(n^2)$	0,7628
Selection Sort	Float	10.000	Pior Caso	$\mathcal{O}(n^2)$	1,0138
Selection Sort	Float	100.000	Melhor Caso	$\mathcal{O}(n^2)$	14,2500
Selection Sort	Float	100.000	Pior Caso	$\mathcal{O}(n^2)$	16,4160
Selection Sort	Object	1.000	Melhor Caso	$\mathcal{O}(n^2)$	1,2094
Selection Sort	Object	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,9771
Selection Sort	Object	10.000	Melhor Caso	$\mathcal{O}(n^2)$	9,5126
Selection Sort	Object	10.000	Pior Caso	$\mathcal{O}(n^2)$	9,9910

Fonte: Carolina Chamas, 2017

O Socket também foi a alternativa mais econômica em um número significativo de vezes, cerca de 27% de casos. Portanto, também foi possível, com base nos dados apresentados na Tabela 11, em que o comportamento pode ser melhor visualizado na Figura 29, extrair as seguintes tendências no que se refere aos casos em que esta forma de comunicação consome menos energia:

- Massa grande de dados, 100.000 elementos, associadas a tipos primitivos Int e Float;

Tabela 11 – Execuções remotas com Socket - Galaxy S

Algoritmos	Tipo dado	Tamanho entrada	Caso	Complexidade	Consumo energia execução com Socket
Bubble Sort	Int	1.000	Melhor Caso	$\mathcal{O}(n)$	0,1638
Bubble Sort	Int	100.000	Melhor Caso	$\mathcal{O}(n)$	5,6551
Bubble Sort	Float	100.000	Melhor Caso	$\mathcal{O}(n)$	4,9361
Heap Sort	Int	100.000	Melhor Caso	$\mathcal{O}(n \log n)$	7,9948
Heap Sort	Int	100.000	Pior Caso	$\mathcal{O}(n \log n)$	7,6000
Heap Sort	Float	100.000	Melhor Caso	$\mathcal{O}(n \log n)$	8,8364
Heap Sort	Float	100.000	Pior Caso	$\mathcal{O}(n \log n)$	5,4158
Bubble Sort	Int	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,1402
Bubble Sort	Float	100.000	Pior Caso	$\mathcal{O}(n^2)$	27,3600
Selection Sort	Int	1.000	Melhor Caso	$\mathcal{O}(n^2)$	0,1326
Selection Sort	Int	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,0744
Selection Sort	Int	10.000	Melhor Caso	$\mathcal{O}(n^2)$	0,8135
Selection Sort	Float	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,1075

Fonte: Carolina Chamas, 2017

As circunstâncias em que o SOAP foi a alternativa mais econômica, conforme apresentadas na Tabela 12, não permitem uma análise mais detalhada sobre situações em que este tipo de comunicação com a rede tende a ser mais econômica, principalmente pelo número limitado de vezes em que essas alternativas foram mais econômicas do que as outras.

Tabela 12 – Execuções remotas com SOAP - Galaxy S

Algoritmos	Tipo dado	Tamanho entrada	Caso	Complexidade	Consumo energia execuções com SOAP
Bubble Sort	Float	10.000	Melhor Caso	$\mathcal{O}(n)$	1,1520
Heap Sort	Float	10.000	Pior Caso	$\mathcal{O}(n \log n)$	1,6172

Fonte: Carolina Chamas, 2017

4.3 Comparação das análises obtidas do celular Galaxy S5 com o celular Galaxy S

Os resultados obtidos com a execução dos experimentos no Galaxy S5 e no Galaxy S são correlacionados e discutidos nesta seção. Para a realização dessa análise, as configurações de execuções que não foram possíveis de serem executadas no Galaxy S são removidas do Galaxy S5 para uma comparação mais precisa. Primeiramente, o comportamento do consumo de energia dos tipos dos dados, dos tamanhos de entrada e complexidades assintóticas dos algoritmos em ambos os celulares são analisados para verificar se existe uma tendência; depois correlaciona-se as circunstâncias em que realizar o offloading é

mais econômico; e então, finalmente, correlaciona-se as tecnologias e arquiteturas de comunicação específicas para a implementação de *computation offloading*.

A Figura 30 apresenta a soma do consumo de energia das execuções locais, por tamanho do dado, tipo de dado e complexidade assintótica. Assim, é possível observar que existe a mesma tendência do consumo de energia dessas variáveis nesses dois celulares. Com relação aos tipos de dados nos dois celulares, Object foi o que mais consumiu energia seguido de Float e então o Int. Quanto ao tamanho do dado, nos dois celulares, as execuções com 100.000 elementos foram as que mais consumiram energia, seguido de 10.000 e então 1.000. Por fim a correlação da complexidade assintótica dos algoritmos também foi a mesma, algoritmos $\mathcal{O}(n^2)$ consumiu mais energia que $\mathcal{O}(n \log n)$ seguido por $\mathcal{O}(n)$.

Figura 30 – Comparação do consumo de energia dos tipos de dados, tamanho e complexidade das execuções locais do Galaxy S5 vs Galaxy S

Fonte: Carolina Chamas, 2017

As Figuras 31, 32 e 33 comparam os resultados obtidos com o S5 e com o S, respectivamente, por tipo de dado, tamanho do dado e complexidade assintótica do algoritmo das execuções remotas com REST, SOAP e Socket. A tecnologia gRPC não foi possível de ser comparada, pois não foi possível realizar suas execuções no Galaxy S.

Figura 31 – Comparação do consumo de energia dos tipos de dado das execuções remotas do Galaxy S5 vs Galaxy S

Fonte: Carolina Chamas, 2017

Figura 32 – Comparação do consumo de energia dos tamanhos dos dados das execuções remotas do Galaxy S5 vs Galaxy S

Fonte: Carolina Chamas, 2017

Dessa forma, é possível observar que existe a mesma tendência do consumo de energia dessas variáveis (tamanho, tipo e complexidade) nesses dois celulares. Com relação aos tipos de dados nos dois celulares, Object foi o que mais consumiu energia seguido de Float e então o Int. Quanto ao tamanho do dado, nos dois celulares, as execuções com 100.000 elementos foram as que mais consumiram energia, seguido de 10.000 e então 1.000. Por fim a correlação da complexidade assintótica dos algoritmos também foi a mesma, algoritmos $\mathcal{O}(n^2)$ consumiu mais energia que $\mathcal{O}(n \log n)$ seguido por $\mathcal{O}(n)$.

Figura 33 – Comparação do consumo de energia das complexidades assintóticas das execuções remotas do Galaxy S5 vs Galaxy S

Fonte: Carolina Chamas, 2017

Dada as observações e os resultados apresentados na Tabela 4 e 9, existem casos em que o consumo de energia das execuções locais e remotas tendem a uma grande proximidade. Esses casos, tanto no Galaxy S5 quanto no Galaxy S, apresentam uma convergência nos resultados, sendo eles:

- Tipo de dado Float;
- Complexidade assintótica $\mathcal{O}(n \log n)$.

Quando se compara os casos em que as execuções remotas foram as mais econômicas, no Galaxy S economizou-se energia em execuções com configurações inferiores ao Galaxy S5:

- majoritariamente as execuções com complexidade $\mathcal{O}(n^2)$ foram mais economicas no servidor, mesmo com entradas pequenas como 1.000, enquanto que no Galaxy S5, somente as execuções $\mathcal{O}(n^2)$ associadas ao tipo de dado Object 1.000;
- massa de dado grande como 100.000 elementos no Galaxy S sempre foi majoritariamente vantajoso a sua execução remota, no galaxy S5 somente quando principalmente associadas com a complexidade quadrática.

Esse resultado mostra a influência da configuração de *hardware* inferior do Galaxy S com relação ao alto processamento com 100.000 ou complexidade $\mathcal{O}(n^2)$.

Com relação às tecnologias de comunicação, ambos os celulares obtiveram uma mesma tendência:

- Geralmente REST é o mais econômico quando associado ao tipo de dado Object;
- Geralmente REST é o mais econômico quando associado com entradas de 1.000 e 10.000 elementos;
- Geralmente Socket é o mais econômico massa de dado maior (100.000 elementos), mas associados a Int e Float;
- SOAP poucas vezes foi o mais econômico.

4.4 Considerações finais

Este capítulo analisou o comportamento do consumo de energia de diferentes tipos de dados, tamanhos de entrada, complexidades assintóticas dos algoritmos e suas influências na escolha da tecnologia de comunicação a ser utilizada na realização da transferência da execução de uma tarefa para um servidor remoto, com os celulares Galaxy S5 e Galaxy S.

De acordo com os resultados observados REST tende a consumir menos energia que as outras, este resultado se deve ao fato que o estilo de comunicação proposto pelo REST tem o objetivo de ser eficiente e simples. Por utilizar HTTP ele possui vantagens no tráfego de dados em detrimento do Socket, que faz uso de conexão TCP padrão a qual abre uma nova conexão para cada par de solicitação e resposta, enquanto que o HTTP utiliza uma única conexão TCP para enviar e receber várias solicitações e respostas. Com

isso, o HTTP permite que vários pedidos usem uma única conexão, tendo como vantagem a diminuição da utilização de recursos reduzindo o congestionamento da rede, da CPU, da memória e também a latência em requisições subsequentes (NIELSEN et al., 1997).

Os resultados evidenciam uma restrição na utilização do REST com grande massa de dados, é possível verificar esta tendência uma vez que os resultados de medidas com tamanho de entrada de 100.000 elementos tiveram, em análise geral, menor consumo de energia com a tecnologia Socket. Krishnamurthy, Mogul e Kristol (1999) apresenta otimizações do HTTP que foram desenvolvidas na versão 1.1, como compactação e envio parcial dos dados pela rede, o que pode evidenciar uma possível hipótese de que o HTTP pode ter restrições de desempenho com grande massa de dados, entretanto, não foi possível encontrar na literatura estudos sobre o consumo de recursos na transferência de grande massas de dados via TCP.

O SOAP possui uma sobrecarga extra por ser baseado em XML o que consome mais energia, primeiramente, pela transformação necessária das informações em XML, o qual possui sua estrutura baseada em árvore, sendo necessária uma lógica mais complexa, diferentemente do JSON, tipo de dado utilizado com REST, que é baseado em chave-valor e, também, pelo custo extra com a largura de banda devido aos metadados necessários na composição de um arquivo XML (PATNI, 2017).

O gRPC, pelo erro obtido em seu processamento no celular Galaxy S, faz grande alocação de memória, possivelmente por seu mecanismo de serialização/desserialização, o que pode resultar em um custo extra de energia.

Com relação aos tipos de dados, Float aloca a mesma quantidade de memória que o tipo de dado Int. No entanto, é possível perceber um aumento de consumo de energia quando o tipo Float é utilizado. Isso pode ser explicado pelo fato de que a alocação de dados Float é diferente, dos 32 bits alocados, 1 bit é para o sinal do número, 8 bits para o expoente e 23 para a mantissa. Assim, quando o tipo de dado Float é armazenado em memória, é necessário normalizá-lo, separar a mantissa, expoente e sinais, portanto existe o aumento de instruções para operar esse tipo de dados (STALLINGS, 2010). Isso demonstra que as operações adicionais utilizadas para lidar com este tipo de dado provocam o aumento do consumo de energia.

Já para o tipo de dados Object, a alocação em memória é maior, e a forma de acesso baseada em *getters* faz com que o número de instruções para acesso dos dados fique ainda maior que no tipo Float. Além disso, este tipo de dado fica armazenado na memória

Heap, a qual possui o acesso às variáveis mais lenta que a memória *Stack*, ratificando a tendência de aumento de consumo energético quando o acesso dos dados requer maior número de instruções (SERSON, 2009).

5 Conclusão

A utilização e o aumento da complexidade dos dispositivos móveis deu início a diversas pesquisas visando diminuir o consumo crescente de energia desses equipamentos, tanto por questões ambientais quanto para a satisfação de seus usuários que não desejam recarregar seus dispositivos em curtos intervalos de tempo. Isso acontece em decorrência da bateria possuir uma evolução historicamente lenta e não acompanhar a evolução dos dispositivos móveis.

Uma técnica que tem sido muito estudada para lidar com o problema do consumo de energia é o *computation offloading*, entretanto a possível redução do consumo de energia com a utilização desta técnica depende da comunicação com a rede, da quantidade de dados a serem transmitidos por ela e também da quantidade de computação a ser realizada. Assim, decidir quando recorrer a essa técnica é uma preocupação fundamental para os arquitetos e desenvolvedores de sistemas móveis.

Embora a tecnologia de comunicação ou o estilo arquitetônico usado para trocar dados com o serviço externo desempenhem um papel importante neste contexto, as abordagens encontradas na literatura não discutem a influência dos protocolos de comunicação e do estilo arquitetônico escolhido no que se refere ao consumo de energia. Portanto, esta dissertação de mestrado teve por objetivo avaliar, por meio de uma pesquisa experimental, o consumo de energia das diferentes técnicas de comunicação cliente-servidor (REST, SOAP, Socket e gRPC) utilizadas em aplicativos móveis, utilizando como apoio algoritmos clássicos de ordenação de diferentes complexidades, diferentes tamanhos de entrada e de tipos de dados.

A contribuição deste projeto de mestrado foi um relatório detalhado da comparação de execuções locais com as tecnologias de comunicação cliente-servidor para aplicativos móveis escritos em Android, para ajudar desenvolvedores e arquitetos a tomarem decisões informadas sobre o uso do *computation offloading* em suas aplicações. Descobrir quais são as escolhas mais econômicas para a comunicação com um servidor externo pode ajudar desenvolvedores a decidir qual tecnologia usar e, portanto, economizar energia.

Dada as análises realizadas, REST tende a ser a tecnologia mais eficiente energeticamente, enquanto que Socket consumiu menos energia para uma massa de dados maior e SOAP e gRPC foram mais econômicos em ocasiões muito raras. Além disso, pode-se ver

que a existência de um tipo de dado mais complexo como Object possui grande influência no consumo de energia, mesmo para uma mesma complexidade de algoritmo.

Descobrir que REST tende a ser o estilo arquitetônico que menos consome energia foi extremamente relevante, desde que a arquitetura de microserviços é uma tendência nas implementações de aplicativos hoje em dia, fornecendo uma entrega rápida de software com serviços independentes e menores (SALAH et al., 2016).

5.1 Contribuições

As contribuições pontuais deste trabalho estão listadas a seguir:

- Realização de experimentos para medir o consumo de energia em dispositivos móveis considerando a combinação de diversos fatores: quantidade de computação, representada pelos algoritmos de ordenação; quantidade de dados, representada pelo tamanho dos vetores e tipos de dados armazenados; e rede, representada pelas tecnologias e estilos arquiteturais de comunicação REST, SOAP, Socket e gRPC.
- Quantificação da influência da complexidade assintótica, do tamanho e dos tipos de dados processados no consumo de energia tanto em execuções locais quanto em execuções remotas.
- Comparação do consumo de energia em dois dispositivos móveis com diferentes configurações, constatando-se que, apesar de apresentarem diferentes consumos de energia, as tendências de maior ou menor consumo são as mesmas nos dois dispositivos considerando-se a influência da complexidade assintótica, quantidade de dados e forma de comunicação utilizada.
- Identificação das duas alternativas de conexão remota para *computation offloading* mais econômicas: REST e Socket.
- Identificação das situações em que as tecnologia de comunicação mais econômicas são mais eficientes energeticamente: REST, com tipo de dado Object e com pequenas quantidades de dados (1.000 e 10.000 elementos); Socket, com grandes quantidades de dados (100.000 elementos) associadas a tipos primitivos (Int e Float).

5.2 Trabalhos publicados

As contribuições científicas originadas diretamente desta pesquisa resultaram em uma publicação:

- Chamas, C.; Eler, M. M.; Cordeiro, D. Comparing REST, SOAP, Socket and gRPC in computation offloading of mobile applications: an energy cost analysis. In Proceedings of the IEEE Latin-American Conference on Communications (LATINCOM 2017). Qualis B2.

5.3 Trabalhos futuros

O projeto de pesquisa aqui apresentado pode ser estendido de diversas forma. Pretende-se, portanto, desenvolver as seguintes atividades como trabalhos futuros:

- Realizar um *benchmark* para verificar o comportamento do protocolo HTTP e TCP com relação a grande massas de dados e utilização de compressão dos dados.
- Replicar os experimentos em dispositivos móveis com diferentes configurações, inclusive de diferentes fabricantes. Acredita-se que com a execução desses experimentos será possível aumentar a massa de dados para mais análises e para identificar se diferentes arquiteturas produzem resultados diferentes dos obtidos com a execução dos experimentos no Galaxy S e S5.
- Executar experimentos usando algoritmos encontrados em aplicações reais, como em jogos e edição de imagens, por exemplo.
- Explorar a utilização de aprendizado de máquina para criar algoritmos que possam decidir dinamicamente se é mais viável energeticamente executar determinada tarefa local ou remotamente. Ainda, espera-se que, com base nos resultados obtidos com a execução de vários experimentos e informações coletadas sobre o perfil de cada aplicação, o algoritmo possa indicar qual forma de comunicação é a mais adequada para que o consumo de energia seja minimizado.
- Implementar um software que seja capaz de realizar modificações automáticas em aplicativos utilizando-se do aprendizado de máquina explorado.

Um estudo aplicando aprendizado de máquina nos resultados obtidos foi iniciado, mostrando que é possível aplicar aprendizado de máquina no contexto de realizar uma análise dinâmica de quando uma estratégia de comunicação é melhor de ser utilizada no sentido de reduzir o consumo de energia dos dispositivos móveis.

O estudo se dá aplicando um rede neural *multiclass*, visando o aprendizado supervisionado, a partir do *dataset* apresentado neste trabalho. O modelo ainda está em fase embrionária, no entanto, algumas características já puderam ser percebidas e aplicadas. Para melhorar a capacidade de aprendizado, foi realizada a troca dos parâmetros, como os nomes dos tipos de dados e das complexidades assintóticas, por um range de valores. Assim, no *dataset*, foi realizada a substituição do termo Int por valores decimais com duas casas de precisão entre os valores 1 e 3. Essa modificação aumentou, significativamente, a precisão da predição.

Uma aplicação para disponibilizar este estudo inicial foi implementada e está disponibilizada em <http://offloadingmlazure.herokuapp.com>. Conforme novas medidas forem realizadas, o *dataset* será incrementado e um novo aprendizado será realizado.

Referências

- AAZAM, M.; HUH, E.-N. Fog computing: The cloud-IoT/IoE middleware paradigm. *IEEE Potentials*, IEEE, v. 35, n. 3, p. 40–44, 2016. Citado na página 17.
- ADEPU, S.; ADLER, R. F. A comparison of performance and preference on mobile devices vs. desktop computers. In: IEEE. *Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), IEEE Annual*. [S.l.]: IEEE, 2016. p. 1–7. Citado na página 20.
- AGUADO, J. M. *Emerging Perspectives on the Mobile Content Evolution*. [S.l.]: IGI Global, 2015. Citado na página 20.
- AKHERFI, K.; GERNDT, M.; HARROUD, H. Mobile cloud computing for computation offloading: Issues and challenges. *Applied Computing and Informatics*, Elsevier, 2016. Citado 2 vezes nas páginas 25 e 40.
- ALBUQUERQUE, F. *TCP/IP internet: programação de sistemas distribuídos: HTML, Javascript e Java*. [S.l.]: Axcel Books, 2001. Citado na página 30.
- BARUCHI, A. *Orquestração de migração massiva de máquinas virtuais baseada em análise cíclica para ambientes de computação na nuvem*. Tese (Doutorado) — Universidade de São Paulo, 2015. Citado na página 17.
- BERNAL, J. F. M. et al. Towards an efficient context-aware system: Problems and suggestions to reduce energy consumption in mobile devices. p. 510–514, 2010. Citado na página 25.
- BOX, D. et al. *Simple object access protocol (SOAP) 1.1*. 2000. <<https://www.w3.org/TR/2000/NOTE-SOAP-20000508/>>. Acessado em: 12/02/2017. Citado 3 vezes nas páginas 31, 32 e 33.
- BUNSE, C. et al. Exploring the energy consumption of data sorting algorithms in embedded and mobile environments. In: IEEE. *Mobile Data Management: Systems, Services and Middleware, 2009. MDM'09. Tenth International Conference on*. [S.l.]: IEEE Computer Society, 2009. p. 600–607. Citado na página 43.
- COHEN, S. *How to stop Waze from gobbling up your iPhone's battery life when you're not using it*. 2017. <<https://mobilesyryp.com/2017/06/24/waze-iphone-battery-saving-ios-app/>>. Acessado em: 05/08/2017. Citado na página 23.
- CORRAL, L. et al. A study of energy-aware implementation techniques: Redistribution of computational jobs in mobile apps. *Sustainable Computing: Informatics and Systems*, Elsevier, v. 7, p. 11–23, 2015. Citado 2 vezes nas páginas 26 e 28.
- CURBERA, F. et al. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *IEEE Internet computing*, IEEE, v. 6, n. 2, p. 86–93, 2002. Citado na página 33.
- DE, D. *Mobile cloud computing: architectures, algorithms and applications*. [S.l.]: CRC Press, 2016. Citado 2 vezes nas páginas 25 e 27.
- DEWAILLY, L. *Building a RESTful Web Service with Spring*. [S.l.]: Packt Publishing Ltd, 2015. Citado na página 34.

DIRKSEN, J. *RESTful Web Services with Scala*. [S.l.]: Packt Publishing Ltd, 2015. Citado na página 34.

FAHIM, A.; MTIBAA, A.; HARRAS, K. A. Making the case for computational offloading in mobile device clouds. In: *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*. New York, NY, USA: ACM, 2013. (MobiCom '13), p. 203–205. ISBN 978-1-4503-1999-7. Citado 2 vezes nas páginas 25 e 40.

FERNANDO, N.; LOKE, S. W.; RAHAYU, W. Mobile cloud computing: A survey. *Future generation computer systems*, Elsevier, v. 29, n. 1, p. 84–106, 2013. Citado 2 vezes nas páginas 26 e 27.

FIELDING, R. Representational state transfer. *Architectural Styles and the Design of Network-based Software Architecture*, p. 76–85, 2000. Citado na página 34.

FISCHER, L. M.; BRISOLARA, L. B. de; MATTOS, J. C. B. de. Sema: An approach based on internal measurement to evaluate energy efficiency of android applications. In: IEEE. *Computing Systems Engineering (SBESC), 2015 Brazilian Symposium on*. [S.l.]: IEEE Computer Society, 2015. p. 48–53. Citado 3 vezes nas páginas 38, 39 e 40.

FUKS, H.; PIMENTEL, M. *Sistemas colaborativos*. [S.l.]: Elsevier Brasil, 2011. Citado na página 36.

GENG, Y. et al. Energy-efficient computation offloading in cellular networks. In: IEEE. *Network Protocols (ICNP), 2015 IEEE 23rd International Conference on*. [S.l.], 2015. p. 145–155. Citado na página 41.

GOMES, D. A. *Web services SOAP em Java: Guia prático para o desenvolvimento de web services Java*. [S.l.]: Novatec, 2ª edição, 2014. Citado na página 30.

GOMES, G. T.; FERTIG, M. R. *Construção de uma base de conhecimento de dados governamentais abertos baseada em ontologia utilizando dados conectados*. [S.l.], 2016. Citado na página 27.

GOOGLE. *gRPC: A high performance, open-source universal RPC framework*. 2017. <<https://grpc.io/>>. Acessado em: 03/09/2017. Citado 2 vezes nas páginas 37 e 38.

GROUP, I. et al. *Introduction to object oriented programming through java*. [S.l.]: Tata McGrawHill, 2007. Citado na página 37.

HALPERN, M.; ZHU, Y.; REDDI, V. J. Mobile CPU's rise to power: Quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction. In: IEEE. *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. [S.l.]: IEEE Computer Society, 2016. p. 64–76. Citado 3 vezes nas páginas 21, 22 e 23.

HAMZAOUI, K. I. et al. Survey on adaptation techniques of energy consumption within a smartphone. In: IEEE. *Science and Information Conference (SAI), 2014*. [S.l.]: IEEE, 2014. p. 247–253. Citado na página 39.

HARPER, A. C.; BURESS, R. V. *Mobile Telephones: Networks, Applications, and Performance*. [S.l.]: Nova Publishers, 2008. Citado na página 47.

JUNIOR, O. A. d. C. *GreenMACC-Uma arquitetura para metaescalonamento verde com provisão de QoS em uma nuvem privada*. Tese (Doutorado) — Universidade de São Paulo, 2014. Citado na página 17.

KAPETANAKIS, K.; PANAGIOTAKIS, S. Efficient energy consumption's measurement on android devices. In: IEEE. *Informatics (PCI), 2012 16th Panhellenic Conference on*. [S.l.]: IEEE, 2012. p. 351–356. Citado na página 40.

KRISHNAMURTHY, B.; MOGUL, J. C.; KRISTOL, D. M. Key differences between http/1.0 and http/1.1. *Computer Networks*, Elsevier, v. 31, n. 11, p. 1737–1751, 1999. Citado na página 83.

KUMAR, K. et al. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, Springer, v. 18, n. 1, p. 129–140, 2013. Citado na página 25.

KUMAR, K.; LU, Y. H. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, v. 43, n. 4, p. 51–56, abr. 2010. ISSN 0018-9162. Citado 4 vezes nas páginas 18, 25, 40 e 42.

KWON, Y.-W.; TILEVICH, E. Energy-efficient and fault-tolerant distributed mobile execution. In: IEEE. *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*. [S.l.]: IEEE, 2012. p. 586–595. Citado na página 26.

KWON, Y.-W.; TILEVICH, E. Reducing the energy consumption of mobile applications behind the scenes. In: *2013 IEEE International Conference on Software Maintenance*. [S.l.: s.n.], 2013. p. 170–179. ISSN 1063-6773. Citado na página 40.

LI, D.; HALFOND, W. G. An investigation into energy-saving programming practices for android smartphone app development. p. 46–53, 2014. Citado na página 24.

LI, R. et al. Cloud-based moving object detection for mobile devices. In: IEEE. *Innovations in Clouds, Internet and Networks (ICIN), 2017 20th Conference on*. [S.l.]: IEEE, 2017. p. 100–102. Citado na página 27.

LIN, K. et al. Energy-accuracy trade-off for continuous mobile device location. p. 285–298, 2010. Citado na página 27.

LINARES-VÁSQUEZ, M. et al. Optimizing energy consumption of guis in android apps: A multi-objective approach. p. 143–154, 2015. Citado na página 24.

MANGLER, J.; BERAN, P. P.; SCHIKUTA, E. On the origin of services using riddl for description, evolution and composition of restful services. In: IEEE COMPUTER SOCIETY. *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. [S.l.]: IEEE Computer Society, 2010. p. 505–508. Citado na página 33.

MANOTAS, I. et al. An empirical study of practitioners' perspectives on green software engineering. In: ACM. *Proceedings of the 38th International Conference on Software Engineering*. [S.l.]: ACM, 2016. p. 237–248. Citado na página 23.

MCMILLAN, M. *Data Structures and Algorithms Using C*. [S.l.]: Cambridge University Press, 2007. Citado na página 43.

- MENNEN, A. *It's Your Call: The Complete Guide to Mobile Phones*. [S.l.]: Relianz Communications, 2005. ISBN 9780975788301. Citado na página 22.
- MURUGESAN, S. Harnessing green it: Principles and practices. *IT professional*, IEEE, v. 10, n. 1, p. 24–33, 2008. Citado na página 17.
- NETO, P. O. A.; SILVA, R. S. *Tecnologias de sistemas distribuídos implementadas em Java: Sockets, RMI, RMI-IIOP e CORBA*. [S.l.], 2008. Citado 2 vezes nas páginas 29 e 30.
- NIELSEN, H. F. et al. Network performance effects of http/1.1, css1, and png. In: ACM. *ACM SIGCOMM Computer Communication Review*. [S.l.], 1997. v. 27, n. 4, p. 155–166. Citado na página 83.
- NIEMEYER, P.; KNUDSEN, J. *Learning Java*. [S.l.]: O'Reilly Media, Inc., 2005. Citado na página 37.
- OLIVEIRA, R. de; FRAGA, J.; MONTEZ, C. *Programação em sistemas distribuídos*. [S.l.], 2002. Citado na página 29.
- OLIVEIRA, R. R. d. *Avaliação de manutenibilidade entre as abordagens de Web Services RESTful e SOAP-WSDL*. Dissertação (Mestrado) — Universidade de São Paulo, 2012. Citado 2 vezes nas páginas 31 e 32.
- ORACLE. *Trail: RMI*. 2016. <<https://docs.oracle.com/javase/tutorial/rmi/>>. Acessado em: 23/02/2017. Citado na página 36.
- PATKARR, M. *Android Battery Killers: 10 Worst Apps That Drain Phone Battery*. 2017. <<http://www.makeuseof.com/tag/android-battery-killers-drain-worst-apps/>>. Acessado em: 05/08/2017. Citado na página 23.
- PATNI, S. *Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS*. [S.l.]: Apress, 2017. Citado na página 83.
- PATRA, B.; ROY, S.; CHOWDHURY, C. A framework for energy efficient and flexible offloading scheme for handheld devices. In: IEEE. *Advanced Networks and Telecommunications Systems (ANTS), 2015 IEEE International Conference on*. [S.l.]: IEEE, 2015. p. 1–6. Citado na página 26.
- PAUL, K.; KUNDU, T. K. Android on mobile devices: An energy perspective. In: IEEE. *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*. [S.l.]: IEEE Computer Society, 2010. p. 2421–2426. Citado na página 23.
- PIERER, M. *Mobile Device Management: Mobility Evaluation in Small and Medium-Sized Enterprises*. [S.l.]: Springer, 2016. Citado na página 20.
- PURUSHOTHAMAN, J. *RESTful Java Web Services*. [S.l.]: Packt Publishing Ltd, 2015. Citado 2 vezes nas páginas 34 e 35.
- SAAB, S. A.; CHEHAB, A.; KAYSSI, A. Energy efficiency in mobile cloud computing: Total offloading selectively works. does selective offloading totally work? In: IEEE. *Energy Aware Computing Systems and Applications (ICEAC), 2013 4th Annual International Conference on*. [S.l.]: IEEE, 2013. p. 164–168. Citado 3 vezes nas páginas 25, 26 e 40.

SAAB, S. A. et al. Partial mobile application offloading to the cloud for energy-efficiency with security measures. *Sustainable Computing: Informatics and Systems*, Elsevier, v. 8, p. 38–46, 2015. Citado na página 23.

SAARINEN, A. et al. Can offloading save energy for popular apps? In: ACM. *Proceedings of the seventh ACM international workshop on Mobility in the evolving internet architecture*. [S.l.]: ACM, 2012. p. 3–10. Citado 2 vezes nas páginas 26 e 40.

SA'ED, T. A. M. Toward green and mobile cloud computing. *Seventh International Conference on Intelligent Computing and Information Systems*, IEEE, v. 3, p. 203–209, 2015. Citado na página 17.

SALAH, T. et al. The evolution of distributed systems towards microservices architecture. In: IEEE. *Internet Technology and Secured Transactions (ICITST), 2016 11th International Conference for*. [S.l.], 2016. p. 318–325. Citado na página 86.

SERSON, R. *Certificação Java 6 - Volume 1 - Teoria*. [S.l.]: BRASPORT, 2009. ISBN 9788574524061. Citado na página 84.

SILBERSCHATZ, A. *Sistemas operacionais com Java*. [S.l.]: Elsevier Brasil, 2008. Citado na página 36.

SILVA, B. d. A. *Um método de otimização da relação desempenho/consumo de energia para arquiteturas multi-cores heterogêneas em FPGA*. Tese (Doutorado) — Universidade de São Paulo, 2016. Citado na página 17.

SNELL, J.; TIDWELL, D.; KULCHENKO, P. *Programming web services with SOAP: building distributed applications*. [S.l.]: O'Reilly Media, Inc., 2001. Citado na página 31.

SPRING. *Understanding HATEOAS*. 2017. <<https://spring.io/understanding/HATEOAS>>. Acessado em: 25/02/2017. Citado na página 35.

STALLINGS, W. *Arquitetura e Organização de Computadores 8a Edição*. [S.l.]: Prentice-Hall, Pearson, 2010. Citado na página 83.

TANENBAUM, A. S.; STEEN, M. V. S. T. et al. *Sistemas distribuídos: princípios e paradigmas*. [S.l.]: Pearson Educación, 2008. Citado na página 36.

TARKOMA, S. et al. *Smartphone energy consumption: modeling and optimization*. [S.l.]: Cambridge University Press, 2014. Citado 3 vezes nas páginas 17, 21 e 22.

TAURION, C. *Cloud Computing-Computação em Nuvem*. [S.l.]: Brasport, 2009. Citado na página 27.

TOH, C. K. *Ad hoc mobile wireless networks: protocols and systems*. [S.l.]: Pearson Education, 2001. Citado 2 vezes nas páginas 22 e 23.

WESTFIELD, B.; GOPALAN, A. Orka: A new technique to profile the energy usage of android applications. In: INSTICC. *Proceedings of the 5th International Conference on Smart Cities and Green ICT Systems - Volume 1: SMARTGREENS*,. [S.l.]: ScitePress, 2016. p. 213–224. ISBN 978-989-758-184-7. Citado 2 vezes nas páginas 20 e 21.

WILKE, C. et al. Energy consumption and efficiency in mobile applications: A user feedback study. In: *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. [S.l.: s.n.], 2013. p. 134–141. Citado na página 23.

YANG, Z. Powertutor-a power monitor for android-based mobile platforms. *EECS, University of Michigan*, retrieved September, v. 2, p. 19, 2012. Citado na página 39.

Apêndice A – Tabulação de todas as execuções realizadas no celular Galaxy S5

Algoritmos	Tipo dado	Tamanho entrada	Caso	Complexidade	Consumo energia execuções locais	Consumo energia execuções com REST	Consumo energia execuções com SOAP	Consumo energia execuções com Socket	Consumo energia execuções com gRPC
Bubble Sort	Int	1.000	Melhor Caso	$\mathcal{O}(n)$	0,0066	0,0971	0,1957	0,4731	0,1535
Bubble Sort	Int	10.000	Melhor Caso	$\mathcal{O}(n)$	0,0562	0,3842	5,2375	0,5723	0,6103
Bubble Sort	Int	100.000	Melhor Caso	$\mathcal{O}(n)$	0,5633	11,3149	5,9985	5,1541	11,2820
Bubble Sort	Float	1.000	Melhor Caso	$\mathcal{O}(n)$	0,0788	0,1221	0,1796	0,2532	0,0909
Bubble Sort	Float	10.000	Melhor Caso	$\mathcal{O}(n)$	0,6806	0,9037	1,0447	0,7419	1,1520
Bubble Sort	Float	100.000	Melhor Caso	$\mathcal{O}(n)$	6,3832	10,3493	7,3783	6,6799	15,2624
Bubble Sort	Object	1.000	Melhor Caso	$\mathcal{O}(n)$	0,1881	0,4674	1,1716	3,0535	0,7943
Bubble Sort	Object	10.000	Melhor Caso	$\mathcal{O}(n)$	2,0480	6,1701	9,5123	25,8733	10,7144
Bubble Sort	Object	100.000	Melhor Caso	$\mathcal{O}(n)$	20,1610	70,5636	104,8919	242,5625	166,3286
Heap Sort	Int	1.000	Pior Caso	$\mathcal{O}(n \log n)$	0,0084	0,0979	0,1281	0,2462	0,1173
Heap Sort	Int	1.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,0120	0,1105	0,1749	0,2819	0,2023
Heap Sort	Int	10.000	Pior Caso	$\mathcal{O}(n \log n)$	0,0852	0,5927	0,6151	0,6963	1,0327
Heap Sort	Int	10.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,0989	0,5654	0,6684	0,5826	1,5551
Heap Sort	Int	100.000	Pior Caso	$\mathcal{O}(n \log n)$	0,9592	8,1453	6,4900	3,9846	7,8089
Heap Sort	Int	100.000	Melhor Caso	$\mathcal{O}(n \log n)$	1,0054	4,9058	7,4100	5,1201	11,1523
Heap Sort	Float	1.000	Pior Caso	$\mathcal{O}(n \log n)$	0,0705	0,1379	0,2245	0,2734	0,1442
Heap Sort	Float	1.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,0846	0,1164	0,1504	0,2004	0,1635
Heap Sort	Float	10.000	Pior Caso	$\mathcal{O}(n \log n)$	0,7362	1,0340	0,7587	0,8494	1,1146
Heap Sort	Float	10.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,7628	0,8458	0,9007	0,6607	2,0516
Heap Sort	Float	100.000	Melhor Caso	$\mathcal{O}(n \log n)$	6,9552	7,9610	12,2928	5,5922	10,8711
Heap Sort	Float	100.000	Pior Caso	$\mathcal{O}(n \log n)$	8,4738	10,3493	7,4396	6,1408	12,7246
Heap Sort	Object	1.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,2069	0,4666	0,9995	0,7987	1,2325
Heap Sort	Object	1.000	Pior Caso	$\mathcal{O}(n \log n)$	0,2785	0,5862	1,3291	0,9457	1,3672
Heap Sort	Object	10.000	Melhor Caso	$\mathcal{O}(n \log n)$	2,3839	3,7863	13,0235	7,1211	19,2446
Heap Sort	Object	10.000	Pior Caso	$\mathcal{O}(n \log n)$	2,6082	53,1084	14,9990	6,7561	11,1523
Heap Sort	Object	100.000	Melhor Caso	$\mathcal{O}(n \log n)$	22,3046	44,8823	133,8276	77,6200	86,2444
Heap Sort	Object	100.000	Pior Caso	$\mathcal{O}(n \log n)$	23,6646	199,2946	129,3667	57,9254	77,6200
Bubble Sort	Int	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,0273	0,1455	0,1888	0,1995	0,1032
Bubble Sort	Int	100.000	Pior Caso	$\mathcal{O}(n^2)$	242,5625	47,9136	36,2710	46,2024	56,5979
Bubble Sort	Int	10.000	Pior Caso	$\mathcal{O}(n^2)$	2,3986	1,6480	1,7885	1,1022	1,8684
Bubble Sort	Float	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,1035	0,1103	0,1515	0,3035	0,1214
Bubble Sort	Float	10.000	Pior Caso	$\mathcal{O}(n^2)$	4,8271	1,7052	1,9241	1,8161	1,8152
Bubble Sort	Float	100.000	Pior Caso	$\mathcal{O}(n^2)$	388,1000	50,7320	51,7467	37,6796	86,2444
Bubble Sort	Object	1.000	Pior Caso	$\mathcal{O}(n^2)$	1,1236	0,6971	1,0281	3,3813	0,9195
Bubble Sort	Object	10.000	Pior Caso	$\mathcal{O}(n^2)$	84,3696	9,7146	15,6492	10,9943	15,6132
Bubble Sort	Object	100.000	Pior Caso	$\mathcal{O}(n^2)$	-	388,1000	388,1000	388,1000	388,1000
Selection Sort	Int	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,0144	0,0808	0,1116	0,2373	0,1345
Selection Sort	Int	1.000	Melhor Caso	$\mathcal{O}(n^2)$	0,0286	0,1073	0,1723	0,3590	0,2247
Selection Sort	Int	10.000	Pior Caso	$\mathcal{O}(n^2)$	0,8761	1,4235	1,1707	1,3544	1,1711
Selection Sort	Int	100.000	Pior Caso	$\mathcal{O}(n^2)$	88,2045	29,5133	23,3795	34,6518	30,5247
Selection Sort	Int	10.000	Melhor Caso	$\mathcal{O}(n^2)$	0,8400	0,7700	1,1201	0,7021	1,5014
Selection Sort	Int	100.000	Melhor Caso	$\mathcal{O}(n^2)$	66,9138	13,4757	16,9476	11,4822	39,6920
Selection Sort	Float	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,0859	0,1101	0,1304	0,2528	0,1669
Selection Sort	Float	1.000	Melhor Caso	$\mathcal{O}(n^2)$	0,0908	0,1225	0,1044	0,3539	0,1566
Selection Sort	Float	10.000	Pior Caso	$\mathcal{O}(n^2)$	2,3450	0,9733	1,0067	1,0259	1,1051
Selection Sort	Float	100.000	Melhor Caso	$\mathcal{O}(n^2)$	121,2813	16,0041	20,2135	22,1771	33,9588
Selection Sort	Float	10.000	Melhor Caso	$\mathcal{O}(n^2)$	1,9137	0,9990	0,9197	0,8871	1,9734
Selection Sort	Float	100.000	Pior Caso	$\mathcal{O}(n^2)$	194,0500	31,8115	31,3616	22,9645	35,6874
Selection Sort	Object	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,5905	0,5746	1,2455	0,7847	1,5713
Selection Sort	Object	1.000	Melhor Caso	$\mathcal{O}(n^2)$	0,6567	0,5887	1,3560	1,1486	1,7529
Selection Sort	Object	10.000	Pior Caso	$\mathcal{O}(n^2)$	44,7808	5,7710	15,4622	10,0284	10,8408
Selection Sort	Object	10.000	Melhor Caso	$\mathcal{O}(n^2)$	47,3293	5,0370	10,3955	7,4065	16,3067
Selection Sort	Object	100.000	Melhor Caso	$\mathcal{O}(n^2)$	-	215,6111	258,7333	242,5625	232,8600
Selection Sort	Object	100.000	Pior Caso	$\mathcal{O}(n^2)$	-	250,3871	388,1000	277,2143	277,2143

Apêndice B – Tabulação de todas as execuções realizadas no celular Galaxy S

Algoritmos	Tipo dado	Tamanho entrada	Caso	Complexidade	Consumo energia execuções locais	Consumo energia execuções com REST	Consumo energia execuções com SOAP	Consumo energia execuções com Socket
Bubble Sort	Int	1.000	Melhor Caso	$\mathcal{O}(n)$	0,0638	0,1810	0,2576	0,1638
Bubble Sort	Int	10.000	Melhor Caso	$\mathcal{O}(n)$	0,7714	1,0527	1,5921	1,4657
Bubble Sort	Int	100.000	Melhor Caso	$\mathcal{O}(n)$	6,4376	18,2851	13,1328	5,6551
Bubble Sort	Float	1.000	Melhor Caso	$\mathcal{O}(n)$	0,1025	0,1465	0,2357	0,1639
Bubble Sort	Float	10.000	Melhor Caso	$\mathcal{O}(n)$	0,8073	1,2899	1,1520	1,2035
Bubble Sort	Float	100.000	Melhor Caso	$\mathcal{O}(n)$	7,2424	11,2610	17,1000	4,9361
Bubble Sort	Object	1.000	Melhor Caso	$\mathcal{O}(n)$	0,4111	1,6157	1,8159	1,7656
Bubble Sort	Object	10.000	Melhor Caso	$\mathcal{O}(n)$	3,3405	8,6052	12,0706	12,7115
Heap Sort	Int	1.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,1121	0,1350	0,2896	0,1755
Heap Sort	Int	1.000	Pior Caso	$\mathcal{O}(n \log n)$	0,0902	0,0570	0,1219	0,0776
Heap Sort	Int	10.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,9365	1,1503	1,2857	1,1958
Heap Sort	Int	10.000	Pior Caso	$\mathcal{O}(n \log n)$	0,9883	1,3473	1,5216	1,4789
Heap Sort	Int	100.000	Melhor Caso	$\mathcal{O}(n \log n)$	8,3330	15,2000	8,0646	7,9948
Heap Sort	Int	100.000	Pior Caso	$\mathcal{O}(n \log n)$	8,4619	10,7294	8,1717	7,6000
Heap Sort	Float	1.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,1172	0,1228	0,3281	0,1583
Heap Sort	Float	1.000	Pior Caso	$\mathcal{O}(n \log n)$	0,1363	0,1094	0,1197	0,1323
Heap Sort	Float	10.000	Melhor Caso	$\mathcal{O}(n \log n)$	1,2732	1,3460	1,4870	1,3813
Heap Sort	Float	10.000	Pior Caso	$\mathcal{O}(n \log n)$	1,5605	1,8411	1,6172	1,6318
Heap Sort	Float	100.000	Melhor Caso	$\mathcal{O}(n \log n)$	13,9909	17,3060	11,1522	8,8364
Heap Sort	Float	100.000	Pior Caso	$\mathcal{O}(n \log n)$	16,4160	12,6101	9,4500	5,4158
Heap Sort	Object	1.000	Melhor Caso	$\mathcal{O}(n \log n)$	0,4815	1,3737	1,7824	1,5213
Heap Sort	Object	1.000	Pior Caso	$\mathcal{O}(n \log n)$	0,5507	1,1726	1,7472	1,4012
Heap Sort	Object	10.000	Melhor Caso	$\mathcal{O}(n \log n)$	4,3660	10,8000	14,0308	12,8250
Heap Sort	Object	10.000	Pior Caso	$\mathcal{O}(n \log n)$	4,2201	6,1102	13,2863	15,7846
Bubble Sort	Int	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,1387	0,1494	0,1607	0,1402
Bubble Sort	Int	10.000	Pior Caso	$\mathcal{O}(n^2)$	7,6000	1,4730	4,7721	2,6048
Bubble Sort	Int	100.000	Pior Caso	$\mathcal{O}(n^2)$	205,2000	19,1914	23,9400	26,7652
Bubble Sort	Float	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,2041	0,1407	0,1827	0,1461
Bubble Sort	Float	10.000	Pior Caso	$\mathcal{O}(n^2)$	10,8715	1,3024	2,0963	1,9133
Bubble Sort	Float	100.000	Pior Caso	$\mathcal{O}(n^2)$	205,2000	39,6000	54,0000	27,3600
Bubble Sort	Object	1.000	Pior Caso	$\mathcal{O}(n^2)$	1,7298	1,3562	1,5275	1,7622
Bubble Sort	Object	10.000	Pior Caso	$\mathcal{O}(n^2)$	46,9029	5,4776	20,2945	10,6751
Selection Sort	Int	1.000	Melhor Caso	$\mathcal{O}(n^2)$	0,1273	0,1608	0,1703	0,1326
Selection Sort	Int	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,1201	0,0886	0,1976	0,0744
Selection Sort	Int	10.000	Pior Caso	$\mathcal{O}(n^2)$	3,0857	1,2595	1,8847	1,3792
Selection Sort	Int	10.000	Melhor Caso	$\mathcal{O}(n^2)$	2,8649	1,0155	0,8860	0,8135
Selection Sort	Int	100.000	Melhor Caso	$\mathcal{O}(n^2)$	143,6400	8,7526	19,3129	19,9500
Selection Sort	Int	100.000	Pior Caso	$\mathcal{O}(n^2)$	205,2000	10,8000	30,7800	41,0400
Selection Sort	Float	1.000	Melhor Caso	$\mathcal{O}(n^2)$	0,1120	0,1033	0,1857	0,1172
Selection Sort	Float	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,1859	0,1690	0,2192	0,1075
Selection Sort	Float	10.000	Melhor Caso	$\mathcal{O}(n^2)$	2,0167	0,7628	1,3959	1,1022
Selection Sort	Float	10.000	Pior Caso	$\mathcal{O}(n^2)$	4,0634	1,0138	1,3807	1,2455
Selection Sort	Float	100.000	Melhor Caso	$\mathcal{O}(n^2)$	111,9273	14,2500	20,5200	23,6769
Selection Sort	Float	100.000	Pior Caso	$\mathcal{O}(n^2)$	205,2000	16,4160	25,6500	26,7652
Selection Sort	Object	1.000	Melhor Caso	$\mathcal{O}(n^2)$	1,0065	1,2094	1,9358	1,3980
Selection Sort	Object	1.000	Pior Caso	$\mathcal{O}(n^2)$	0,9544	0,9771	1,9000	1,1099
Selection Sort	Object	10.000	Melhor Caso	$\mathcal{O}(n^2)$	14,8696	9,5126	14,1517	10,8818
Selection Sort	Object	10.000	Pior Caso	$\mathcal{O}(n^2)$	17,8435	9,9910	17,1000	10,7372