

UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science
Software Engineering Curriculum

Oluwaseni Toluwaleke Joshua

A Framework for Energy-efficient Mobile Cloud Offloading

Master's Thesis (30 ECTS)

Supervisor: Satish Narayana Srirama, Ph.D.

Chii Chang, Ph.D.

Tartu, 2015

A Framework for Energy-efficient Mobile Cloud Offloading

Abstract

Emerging smartphone technologies has experienced a geometric increase and is currently still on the rise. People use the smartphone for their day-to-day activities such as sending emails, sharing photos and videos through various peer-to-peer social network hubs and so on. In the last few years, the smartphone has experienced massive technological advancements and innovation with respect to its processing capabilities and can now be used to perform complex, resource-intensive tasks in advanced applications like video editing and processing, and object recognition. Although most smartphones have been greatly augmented to handle advanced applications with complex computational needs, they are still limited in terms of their energy resources i.e. battery life. Battery technology has not evolved as rapidly as other areas of the smartphone and so the execution of computational-intensive tasks would cause its rapid depletion; evidenced by the need to constantly charge the device battery.

Many techniques have been proffered to maximize energy conservation on mobile devices. Some of which are slowing down the CPU, or shutting off the screen when idle. Among these, the most notable technique for conserving smartphone energy is computation offloading. This basically involves the transfer of the processing of certain tasks from a resource-constrained smartphone to a remote, resource-rich device thereby facilitating energy conservation on the smartphone. This is a fairly large research area and numerous contributions have been made towards advancement in this field. However, much work is yet to be done with regards to energy conservation through offloading during recurrent resource-intensive processing.

In this research study we aim to reduce energy consumption during continuous, energy-intensive processing. We consider context-awareness in proposing a scheduling model that could potentially minimize the speedy depletion of mobile device energy thus achieving our aim. We propose a

service-oriented framework towards enabling energy-optimal task execution through a task scheduling offload algorithm. We develop a proof-of-concept prototype on an Android device to demonstrate and evaluate the framework's energy conserving capabilities.

Keywords: Computation Offloading, Cloud Computing, Task Scheduling, Service-oriented Architecture, Android, Smartphones

Raamistik mobiilse koormuse jaotamiseks pilve

Lühikokkuvõte:

Nutitelefonide tehnoloogiline areng on kogenud eksponentsiaalset kasvu ning on jätkuvalt tõusuteel. Nutitelefone kasutatakse igapäevaselt näiteks e-mailide saatmiseks, piltide ja videode jagamiseks seadmete vahel kasutades sotsiaalvõrgustike rakendusi. Viimastel aastatel on nutitelefonide tehniline võimekus kasvanud hüppeliselt, mis on teinud võimalikuks keerukate ja ressursimahukate ülesannete täitmise keerukates rakendustes, näiteks videotöötlus ja –analüüs ning objektide tuvastus. Kuigi enamus nutitelefonide võimekus on piisav, et käsitleda keerukaid rakendusi, on siiski piiratud nende energia varud, ehk aku kestvus. Märk sellest on pidev vajadus seadme akut laadida.

Välja on pakutuid mitmeid energia säästmise meetodeid nutitelefonides. Osad aeglustavad protsessorit või lülitavad välja ekraani kui seade on ooterežiimil. Üks märkimisväärsematest meetoditest on arvutuskoormuse jaotamine. See hõlmab teatud ülesannete töötamise üleviimist piiratud ressursidega nutitefonist kaugesse ressursirikkasse seadmesse hõlbustades seega nutitelefoni energia kokkuhoidmist. Tegemist on suure uurimisvaldkonnaga, millesse on palju panustatud. Sellele vaatamata on veel palju tööd vaja teha seoses energia säästmisega läbi arvutusvõimsuse koormuse jaotamise korduva ressursimahuka töötlemise ajal.

Antud lõputöö eesmärgiks on välja töötada raamistik, mis vähendab energia tarbimist korduva energiamahuka töötlemise ajal. Algoritmis võetakse arvesse olemasolevaid ressursse, mis aitab minimeerida energia kasutust seadmes. Samuti arendatakse välja *proof-of-concept* Android rakendus, et demonstreerida ja hinnata raamistiku energiasäästu võimekust.

Võtmesõnad: Arvutuskoormuse jaotamine, Pilvandmetöötlus, Teenusele-orienteeritud arhitektuur, Android, Nutitelefoniid

Table of Contents

List of Figures	viii
List of Tables	ix
1 Introduction.....	2
1.1 Preamble.....	2
1.2 Motivation.....	3
1.3 Problem Statement	5
1.3.1 Research Problem.....	5
1.4 Contribution	6
1.5 Thesis Outline	6
2 Related Work	7
2.1 Mobile Cloud Middleware	8
2.2 Energy-efficient Task Scheduling in Mobile Cloud Computing	9
2.3 Energy-efficient Multisite Offloading Algorithm for Mobile Devices.....	9
2.4 Energy-optimal Mobile Application Execution for Resource-poor mobile devices.....	10
2.5 Energy-efficient Scheduling for Collaborative Execution in Mobile Cloud Computing.....	11
2.6 Energy-efficient Computation Offloading model for Mobile Phone Environment	12
2.7 Comparative Analysis	13
2.8 Summary	16
3 System Design	17
3.1 System Description and Architecture.....	18
3.1.1 Architecture Overview	18
3.1.2 Context-aware Offload Scheduling Algorithm	20
3.1.2.1 Prediction Model	22
3.1.2.1.1 Prediction Model Development.....	22
3.1.2.2 Scheduling Algorithm.....	25
3.2 Summary	26
4 Implementation	27
4.1 Mobile Systems.....	27
4.1.1 Smartphones	27
4.1.2 Mobile Operating Systems	28
4.1.2.1 Android Operating System.....	29
4.2 Cloud Computing	29

4.2.1	Cloud Computing Services	31
4.2.2.1	Amazon Elastic Compute Cloud.....	32
4.2.2.1.1	Amazon Elastic Compute Cloud Instance types	33
4.2.2.1.2	Amazon Elastic Compute Cloud Features.....	33
4.3	Service-Oriented Architecture	34
4.4	Arduino	35
4.4.1	Arduino Board	35
4.4.2	Arduino Development Environment.....	36
4.4.2.1	Arduino Sketches	36
4.4.3	Shields.....	37
4.4.3.1	RedBearLab BLE Shield.....	38
4.4.3.2	TinkerKit Mega Sensor Shield.....	39
4.4.4	Sensors	39
4.5	Machine Learning	41
4.5.1	Supervised Learning	41
4.5.1.1	Linear Regression	42
4.5.2	Unsupervised Learning	43
4.5.3	Reinforcement Learning	43
4.6	State Machine.....	43
4.7	Servlet Container	44
4.8	Implementation Overview	45
4.8.1	Context Sensor Setup.....	47
4.8.2	Configuring the Prototype.....	48
4.8.2.1	Android Service Setup	50
4.8.2.2	Implementing the State Machine	52
4.8.2.3	Implementing the Scheduling Algorithm.....	53
4.8.2.4	Servlet Container Configuration.....	54
4.9	Summary	55
5	Experimentation.....	57
5.1	Use Case and Setting.....	57
5.1.1	Energy-Conservation Evaluation.....	57

5.2	Results.....	58
5.2.1	Test Case 1.....	58
5.3	Discussion.....	59
6	Conclusion and Future Works.....	61
	Bibliography.....	64

List of Figures

Figure 1: The System Design.....	19
Figure 2: Cloud Computing Models	30
Figure 3: Arduino Mega ADK Board	36
Figure 4: Arduino DE with a Sketch.....	37
Figure 5: RedBearLab BLE Shield	38
Figure 6: TinkerKit Mega Sensor Shield	39
Figure 7: TinkerKit Thermistor Module	40
Figure 8: Temperature Sensor Node	40
Figure 9: Stately.js Sample State Machine	44
Figure 10: Temperature Parsing Scenario.....	46
Figure 11: Extracting sensor data Arduino code snippet	48
Figure 12: Class diagram of prototype.....	49
Figure 13: Bluetooth Scanning	51
Figure 14: BroadcastUpdate function	52
Figure 15: State Machine of Prototype	53
Figure 16: Task Scheduling Algorithm Source Code	54
Figure 17: Prototype Use Case Diagram	55
Figure 22: CPU consumption rate for an Offload Decision	59

List of Tables

Table 1: Comparison Table.....	14
Table 2: Single Training data set tuple	23
Table 3: Transformed training dataset	24
Table 4: CPU Consumption for offloading.....	58
Table 5: CPU consumption for local computation	59

Acknowledgements

I would like to thank Almighty God without whom none of this would be possible. Secondly, I would like to show appreciation to and acknowledge my parents for their continued love, guidance and support. I also thank Prof. Marlon Dumas, the International Masters in Software Engineering program coordinator, my primary supervisor Assoc. Prof. Dr. Satish Narayana Srirama and co-supervisor Dr. Chii Chang for their continued guidance and willingness to help throughout this Master's program and during the writing of this document, irrespective of their schedule. Finally, I thank the University of Tartu for the opportunity to learn at the best University in the Baltics.

1 Introduction

1.1 Preamble

The advent of the smartphone has made way for numerous technological advancements and innovations with respect to information technology. It has facilitated the performance of high-level functionalities such as capturing and storing images taken from the device to the cloud, GPS tracking and so on. A recent survey by (Cisco, 2015) has shown that there are more mobile-connected devices than the population of people in the world today and by 2019 there will be almost 1.5 mobile devices per capita. They forecasted that due to the massive usage of smartphones, the average global mobile connection will surpass 2 Mbps by 2016 and that there will be an increase in mobile device traffic on wireless networks i.e. Wi-Fi, 3G/4G/5G mobile Internet.

Smartphones have experienced a lot of innovative, technological advancements from heavy obtuse looking devices with monophonic ringtones to slim, sleek, lightweight devices. Today's mobile phones have their computational power far greater than desktop computers used in several years ago. Nowadays, people can perform various activities with their smartphones including sending email, controlling one's home thermostat, surveillance, GPS navigation, augmented reality visualization, gaming and so on. These are just a few of the numerous, complex applications that mobile devices have been outfitted to handle. These functionalities are a far cry from the initial goal of cell phones i.e. connecting people in different locations through phone calls. All this is possible because smartphones have been greatly augmented in terms of processing power and computational abilities. However, there is an opportunity cost depicted by the adverse effects these complex processes have on the battery life. Today's consumers largely desire smartphones that have exceptional computation power, are lightweight, slim, and fast in order to be able to handle their advanced needs. As a result of this, more applications with advanced computational

requirements are launched on the smartphone. The need to have smaller, faster smartphones with advanced processing capabilities has created a tradeoff; with the battery life of these devices taking the brunt of it. Most smartphones run on battery power which is finite in nature and the running of these complex applications would only consume more power thus reducing the battery life. This tends to be a problem as the usability of smartphones is dependent on the amount of battery life readily available. In situations where some applications are continuously generating heavy data through intense computation, the life of the battery would potentially face rapid exhaustion. Although, smartphones have undergone a lot of improvements in recent years and it has become able to execute complex processes and functionalities, their batteries have not experienced similar tandem growth and as such cannot handle the ever-increasing power consumption rate of these devices. This is very evident in smartphones such as the Google Nexus and Apple iPhone series where users need to constantly charge their phones. In this sense, a concept known as offloading (Li, Wang, & Xu, 2001) has been proposed as a viable solution to these energy conservation problems in today's smartphones.

1.2 Motivation

In layman terms, our prime motivation for this research study is that we desire mobile devices that can adequately handle complex computation and data processing without suffering ghastly losses in battery life. In other words, we aim to reduce the energy consumption during the execution of complex tasks by advanced applications. Several research efforts (Li et al., 2001), (Xian, Lu, & Li, 2007) and (G. Chen et al., 2004) have been proposed with the aim of implementing a solution to reduce the battery consumption of smartphones. As mentioned earlier, one of the major solutions for smartphone's energy conservation that has been considered in these research studies is computation offloading. Computation offloading is a process that allows a resource-constrained device to transfer and executes whole or a part of an application or its tasks on a remote, resource-rich server and receives the processing result from it. It is sometimes referred to as cyber foraging (Balan, Flinn, Satyanarayanan, Sinnamohideen, & Yang, 2002). Computation offloading is widely applicable in many instances such as in context-aware systems in which several data streams from

numerous sources i.e. motion sensors, thermostats etc. needs to be processed in order to extract information about an individual's surroundings. Processing such data on power-constrained smartphone could be crucial and so these devices processing and computation can be improved by utilizing the offloading technique. Computation Offloading is usually performed at certain levels of granularity. This means that offloading can occur at the whole application level (Chun & Maniatis, 2009), at the class level (Geoffray, Thomas, & Folliot, 2006), method level (Cuervo & Balasubramanian, 2010) or object level (Niu, Song, & Liu, 2013). Several methods are used to enact the process of offloading. Most of these methods involve the use of algorithms that decide when one should offload or not. These algorithms also enable the partitioning of the application according to the afore-mentioned granularities. For example, EETS (Yao, Yu, Jin, & Zhou, 2013) employed a cost graph for determining offload suitability at the task level. Other kinds of these offloading algorithms will be examined in the Chapter 2 of this research study. Offloading in the context of conserving energy for mobile devices is very successful as reported by the evaluations of several research studies like the above mentioned EETS. In our cursory look at these works, most of them considered an algorithm that for energy conservation purposes, offloaded once or when prompted, either to a single server or a combination of different servers as in the case of EMSO (Niu et al., 2013). Although most solutions are impressive and effective with regards to energy conservation, little or no consideration has been towards maximizing energy conservation during continuous processing. We established in earlier sections that the battery of smartphones are limited and that they quickly get exhausted due to heavy processing from the advanced applications. In this study, we consider a scenario where a smartphone is constantly doing processing as in the case of parsing data from sensors such as proximity, temperature and tracking sensors within a certain vicinity (e.g. in a real-time augmented reality scenario). Upon that premise, we proffer a service-oriented framework that incorporates a context-aware offloading scheme, which facilitates energy-optimal execution of such recurrent processing tasks either through offloading or local computation.

1.3 Problem Statement

In this section we outline and analyze the problems involved in minimizing the energy depletion of smartphones while continuously handling data or content from numerous sources. We also present several research questions that we aim to resolve within this thesis's context.

1.3.1 Research Problem

Earlier in this chapter, we examined the advent of smartphones and how their resource-limited nature hinders users from continuous usage of advanced applications without needing to worry about the battery level. We also made mention of offloading as a possible solution to conserving the energy of smartphones and gave instances of its effectiveness through various research studies. In this study, we look at the issue of energy conservation within the context of applications that continuously retrieve data from external sources such as sensors. This usually occurs when the smartphone user comes within range of the source(s). The data retrieval is done mostly through a common protocol such as Bluetooth Low Energy (BLE) (DeCuir, 2014). The BLE protocol is the preferred data transfer protocol for resource-constrained devices as it provides reduced power consumption over a similar communication range as the Bluetooth protocol. The constant processing of data of varying size tend to take its toll on the battery and so a lightweight, resource-conserving method of saving battery life during such computation-intensive processes is imperative. In the next chapter, we will examine similar research works aimed at conserving the energy of the mobile device.

In this paper, we propose a framework that leverages a context-aware scheduling algorithm within the context of continuous offloading to deal with the research questions as given below:

- How do we maximise energy conservation on mobile devices during continuous execution of complex processing tasks?
- What factors should be considered in making the decision to offload computation to the cloud with the goal of energy conservation in mind?

1.4 Our Contribution

In this research study, the primary aim is to foster energy conservation on mobile devices while continuously processing data for complex applications. The proposed contributions are listed below:

- Develop a scheduling scheme that determines the most energy-optimal and suitable alternative in offload decision-making.
- Develop a service-oriented framework that leverages the scheduling scheme in conserving energy for mobile devices within a continuous processing scenario.

1.5 Thesis Outline

This thesis is outlined as follows:

Chapter 2 reviews a number of related works in mobile cloud-based computational offloading. A comparison table is provided to summarize the efforts and the gaps.

In Chapter 3, we describe the detail of the design, architecture and major components of the proposed framework.

Chapter 4 provides the detail of the procedures and steps taken in implementing and developing our proof-of-concept prototype. It also describes the various tools, frameworks, technologies and other applications that we employed in implementing and finalising our approach.

The experimental evaluations of the proposed approach with regards to energy conservation during resource-intensive processing is presented in **Chapter 5**. We also present validations of the prediction model in our proposed scheduling algorithm.

This thesis is concluded in **Chapter 6** together with future research direction.

2 Related Work

Substantial research has been done with respect to reducing energy consumption on mobile devices by using scheduling algorithms that foster efficient computation offloading. One of the major motivations for this research direction is the need to conserve battery life of mobile devices which in today's smartphones gets easily depleted due to intensive usage i.e. surfing the net, browsing, and playing games.

As mentioned in Chapter 1, one salient method of conserving mobile device energy is through computation offloading. In itself, computation offloading is the act of transferring part of an application's code or a specified task to a remote resource-rich location for computation. This is done usually when the task has been deemed too resource-intensive to be performed locally. There are various forms of computation offloading such as *whole-application offloading* (Chun & Maniatis, 2009), *task offloading* (E. Chen, Ogata, & Horikawa, 2012) and *code offloading* (Geoffray et al., 2006).

- *Whole-application offloading* is self-explanatory as it involves the situation where the processing of a whole application is offloaded to a server.
- *Task offloading* is much different in that it involves the offloading of a specific task/process of an application or a series of tasks.
- *Code offloading* is slightly similar in the sense that it involves offloading a particular part of an application's code for example an object, function or class to a remote server on the cloud.

This brings us to the issue of cloud computing and their interaction with mobiles. Cloud computing is defined simply as providing vast computing resources as a utility. It can also be described as a platform that provides network-based services based on the quantity needed at that time. The advent of cloud computing has created a new research impetus in various fields of mobile computing augmentation leading to the development of the mobile cloud computing paradigm. Comprehensively, Mobile Cloud Computing (MCC) (Fernando, Loke, & Rahayu, 2013) is a rich mobile computing technology that uses the unified elastic resources of various cloud and network technologies to provide unrestricted functionality, storage and mobility for a vast number of mobile devices anywhere, at any point in time through the Internet; regardless of differing environments and platforms through the pay as you use principle. In this Chapter, we review a number of mobile cloud computing frameworks that have been used to conserve mobile device energy through offloading. First, we review the frameworks singularly and then we present a comparative analysis of them in tandem with our proposed solution.

2.1 Mobile Cloud Middleware

Mobile Cloud Middleware (Flores & Srirama, 2013) is mainly focusing on delegation and composition of tasks from mobile devices to several clouds in order to enhance the processing capabilities and conserve the resources (i.e. battery life) of the mobile device. MCM manages asynchronous task delegation of mobile devices across multiple clouds. The authors justified that using MCM to offload to several clouds can help save more energy on the device. This eliminates the need of recurrent task delegation to a cloud server by leveraging a composition mechanism. This mechanism is called—*service composition*. Its role is to compose several tasks together and offload them to the various clouds in a single invocation from the mobile device. A demonstration of MCM and its inherent service composition feature was depicted using a hybrid Android mobile cloud application. The authors verified the middleware's scalability through simulating different mobile loads and assessing how the middleware handles the heavy loads. It was concluded that MCM is able to effectively handle large loads in addition to dealing with multi-cloud processes from a mobile application towards improving the quality of service for mobile devices.

2.2 Energy-efficient Task Scheduling in Mobile Cloud Computing

Managing data transmission from mobile devices to the cloud while executing mobile tasks is important in order to augment and fully utilise the processing capabilities of the mobile device and conserve its energy. The authors (Yao et al., 2013) proposed an Energy-Efficient Task Scheduling (EETS) algorithm that determined what type of task was suitable for offloading to the cloud and if the task should be offloaded or executed locally on the mobile device. Their proposed task scheduling algorithm takes the data input/output locations, data size, compression ratio, and available bandwidth into consideration and employed a cost graph in order to denote if it was advisable or not to offload the computation of a particular task to the cloud. The cost graph depicted empirical, visible evidence that showed how the offload/no offload action would help saving energy of the mobile device. It was highly dependent on where the input data came from and where the output data would be stored (i.e. mobile device or cloud server) in order to calculate the energy consumption difference between offloading a computation and executing on the device. The authors performed an evaluation of the algorithm using an Android mobile device. Their results showed that the algorithm was efficient in that it had an accuracy of 99% when compared with a function that showed the energy consumption costs of offloading a task to local execution.

2.3 Energy-efficient Multisite Offloading Algorithm for Mobile Devices

The authors (Niu et al., 2013) of this study, surmised that it is much more beneficial to employ computation offloading to send parts of an application to multiple remote servers in order to reduce energy consumption and enhance execution times of tasks on smartphones. They proposed a multisite partitioning algorithm known as Energy-Efficient Multisite Offloading (EMSO) algorithm that used the multiway graph partitioning based algorithm to split an application into smaller chunks. The authors partitioned applications at the object level. These objects that can be

offloaded to one or more clouds for distributed execution in order to facilitate an improvement in energy conservation. The authors emphasised that simply using an algorithm or scheduling policy to offload computation to the cloud does not explicitly reduce energy consumption. It is because several issues influenced the computation offloading and adversely affects its efficiency and performance. Issues such as inconsistency in wireless network bandwidths and improper partitioning of the application. i.e. applications partitioned at the class level. The authors' aim was to develop an offloading algorithm that took the mitigation of these challenges into consideration. In identifying which parts of an application should be partitioned and offloaded to the cloud, EMSO restructured the applications as a Weight Object Relation Graph (WORG), which was constructed using static analysis and offline processing (Niu et al., 2013). EMSO used the WORG as an input to the multiway graph partitioning based algorithm to resolve the multiway partitioning problem formulated as an Integer Linear Programming problem. The result depicted the ideal partitioning to which the applications could be split for offloading and subsequent execution on several clouds. The authors made experiments to determine how effective EMSO was in reducing energy consumption and enhancing execution times during offloading/partitioning on smartphones. Results showed that EMSO was capable of reducing energy consumption and execution time whilst partitioning and offloading the application from mobile devices to multiple clouds. It adapted properly to bandwidth fluctuations of the wireless network and ultimately performed comparatively better than the No Application Partitioning and Static Application Partitioning algorithm (Ou, Yang, & Liotta, 2006).

2.4 Energy-Optimal Mobile Application Execution for Resource-poor Mobile Devices

As implied earlier in this text, although mobile applications and devices nowadays have become advanced that they are able to handle complex computational tasks, they still lack in terms of battery power. As such, several research like the ones mentioned above has suggested methods to the resolution of the effects of battery-life shortage. (Wen, Zhang, & Luo, 2012) focus on a scheme to employ cloud computing to augment the processing and storage capabilities of mobile devices

through an energy-efficient mobile application execution management scheme. Since, a mobile application can be executed either on the mobile device or on the cloud, the authors' scheme proffered a way to determine on which platform (i.e. mobile or cloud) would executing a certain application yield the most energy benefits. The authors developed an energy-aware optimization system that designated execution to the mobile and/or cloud in order to reduce the device's energy consumption within a certain time limit. In designing this framework for energy conservation, they implemented two scheduling approaches;

- the first concerned optimally scheduling the mobile device's clock frequency as that helped maximise energy conservation whilst executing the application on the mobile device and
- the second was scheduling the data transmission rate through a wireless channel in order to maximise the conservation of transmission energy when the mobile applications are executed in the cloud.

These two approaches constituted the scheduling problems to which the authors formulated solutions. The authors applied these solutions in order to formulate the optimal application execution policy that determined which executions (i.e. on mobile or on cloud) for a particular mobile application was more energy efficient. Subsequently, they ran numerical simulations that denoted that their execution management policy saved up to 13 times more energy for the mobile device by regulating the offloading of mobile applications to the cloud.

2.5 Energy-efficient Scheduling for Collaborative Execution in Mobile Cloud Computing

This approach (Zhang, Wen, & Wu, 2013) dealt with the mobile application in a more granular way; converse to (Wen et al., 2012) that focused on regulating the offloading of the whole application to the cloud. This study is virtually an extension of EOMAE which we addressed in the preceding section. It looked at applications as a series of sequential tasks and focused on a scheme that facilitated the conservation of mobile device energy through the offloading of tasks from the mobile device to the cloud. This was the aim of the authors; collaborative execution of

tasks between the mobile device and cloud for energy conservation under a time deadline. This paper proposed the development of an energy-conscious task scheduling algorithm (EESP) under a Markovian Stochastic channel that managed the utilization of the mobile device energy to that effect. The scheduling issue was expressed mathematically as a constrained stochastic shortest path problem on a directed acyclic graph converse to EMSO that modelled the scheduling problem as a 0-1 Integer Linear Programming problem. In order to procure the optimal task scheduling policy for collaborative execution, they implemented a Lagrangian Relaxation Based Aggregated Cost (LARAC) algorithm hence resolving the shortest path problem. Summarily, the authors performed numerical simulations in order to evaluate the policy's performance. The results of their evaluations depicted the effectiveness of the one-climb policy in energy conservation for smartphones. The one-climb policy simply entails the one time offload of tasks to the cloud for collaborative execution.

2.6 Energy-efficient computation offloading model for Mobile phone environment

The focus of this study (Fekete, Csorba, Forstner, Feher, & Vajk, 2012) was mainly on implementing a custom task scheduling paradigm for offloading computation to a remote, resource-rich resource in order to enable energy conservation on the mobile device and present a measurement framework to measure the amount of energy consumed during computation. The authors proposed a novel, energy-efficient job scheduling (EEJS) scheme with the aim of maximising energy conservation during computation execution on a smartphone. The scheduling scheme is based on the classical makespan minimization scheduling problem. They intend for the model to be able to also decide which particular task should be offloaded to the cloud and which should not considering factors like job complexity and whether 3G or Wi-Fi is the communication medium for task offloading. EEJS employs measures to schedule transmission jobs and normal computation jobs separately. They affirm that this is key to energy saving. They modelled the scheduling problem as a graph and proposed implementing an 0-1 ILP solver to solve it. They denoted that a series of jobs that consume a minute amount of energy would be produced as a

result. In order to assess the efficacy of EEJS in energy conservation the authors developed a measurement system that analysed the energy consumption distribution on the mobile phone. They also implemented a logging system into the mobile device that denoted which activity caused which energy consumption level. However, a possible drawback with this study is that, although the authors presented reasonable evidence as to the possibility of reducing energy consumption with their methods, they did not provide actual evidence that EEJS reduced energy consumption during the execution of a certain task. The measurement results they presented merely displayed the baseline energy consumption levels in various states.

2.7 Comparative Analysis

This section presents a summary of the above studies and compares each of them with respect to their modes of task scheduling, offloading techniques and ultimately how they distinctly enhanced energy conservation on smartphones. These are depicted in the table below.

	Offloading Mode			Strategy				Model	Strategy for energy efficient offloading?	Strategy for continuous offloading?
	Application	Code	Task	Scheduling		Partitioning	profiling			
				Policy-based	Context-based					
MCM	No	No	Yes	Yes	No	No	No	Composite task offloading	No	No
EETS	No	No	Yes	No	Yes	No	Yes	Cost graph	Yes	No
EMSO	Yes	No	No	Yes	No	Yes	No	Multiway graph partitioning	Yes	No
EOMAE	Yes	No	No	No	Yes	No	No	Custom execution policy based on optimal mobile clock frequency and data transmission scheduling problems.	Yes	No
EESP	No	No	Yes	Yes	No	No	No	Custom task scheduling policy using the LARAC (5) algorithm	Yes	No
EEJS	No	No	Yes	No	Yes	Yes	No	Custom Job scheduling model.	Yes	No
Our Implementation	No	No	Yes	No	Yes	No	No	Context-aware task scheduling algorithm	Yes	Yes

Table 1: Comparison Table

We examine the concepts and measures in the table which we considered in comparing between these studies.

- **Offloading Mode:** This describes what system/application object was offloaded i.e. at what level was the application and offloaded? The whole application, code or task level?

- **Strategy:** This concerns what methods were used in delineating the application into offloading objects i.e. code, task, and whole application. It also answers the question of if the offloading mode was profiled or partitioned?
 - ◆ **Scheduling:** This defines what type of algorithms were used to leverage the offloading model for the given granularity (whole application, code or task).
 - **Policy-based Algorithms:** Did the solution follow a set of static rules in determining offloading?
 - **Context-based Algorithms:** Did the solution follow an offloading process based on current state of mobile device resources?
 - ◆ **Partitioning:** Was the offloading mode partitioned into smaller divisions?
 - ◆ **Profiling:** Were the offloading objects profiled according to their type and complexity?
- **Model:** This answers the question, what method or techniques were used in offloading? What measures determined the suitability of offloading?
- **Strategy for energy efficient offloading:** Was the solution strictly concerned with maximising energy conservation for the mobile device through offloading?
- **Strategy for continuous offloading:** Did the solution consider an offloading mechanism during continuous processing of tasks?

From the above table, one can infer that majority of the considered research studies did not implement any strategy for continuous offloading. They rather focused on conserving energy of a mobile device within a traditional offloading scenario. They achieved this, using a range of models and approaches however only EETS, EOMAE and EEJS implemented a context-aware offload scheduling scheme in semblance to our approach.

2.8 Summary

A lot of the reviewed approaches are similar to the approach we followed. However, our approach is an extension in that we leverage task scheduling in our framework to conserve energy whilst continuously provisioning content (i.e. from sensors and web services) which in order to process, may require offloading to a server on the cloud or local mobile execution. This is the distinction of our approach from others. This chapter has been a review of related work by other researchers with respect to energy efficient offloading techniques. We also presented a comparison between the reviewed solutions and the nuances of their approach. In the next chapter, we present an overview of the architecture of our proposed service-oriented offloading framework.

3 System Design

This chapter introduces the structure and design of our proposed service-oriented mobile offloading framework.

As we have mentioned in the introduction, although smartphones of today have been greatly augmented in terms of their processing capabilities, their battery power has not experienced similar growth. Running resource-intensive tasks on such smartphones ends up depleting the battery power quickly. One of the reasons of this speedy battery depletion is that some mobile applications are constantly running in the background especially on the Android OS (Nimodia & Deshmukh, 2012). As time goes on, these applications i.e. YouTube, Maps, Mail, Facebook, get much heavier to handle due to increase in data-intensive and graphic-intensive tasks which is as a result of regular updates and such. This tends to bring about rapid battery consumption. Our main objective is to minimize energy consumption during such heavy, recurrent data processing through the employment of an energy optimal execution scheduling algorithm.

We address the challenges in developing such a framework. These issues will mainly center on hardware constraints and communication issues. With respect to hardware constraints, since our target environment is the mobile environment, we would have to deal with the constraints set by the mobile platform in accessing specific functions and APIs. Compatibility has to do with the ability of the framework components to interact with one another efficiently. In the subsequent sections of these chapter, we elaborate on the features of the framework and this includes the architecture, the offloading algorithmic model and so on.

3.1 System Description and Architecture

The system is a service-oriented framework that facilitates energy conservation of mobile devices using a task scheduling algorithm that proffers energy-efficient continuous mobile offloading. Within the context of this research, due to time and resource constraints, we limited our scope to conserving energy of the mobile device while processing or parsing data from sensors specifically temperature sensors. The main requirements of the system are listed below:

- **Data Retrieval:** The framework should be able to retrieve context information from external, small, physical computing devices such as sensors. It should extract data over a specified network protocol.
- **Task Execution Scheduling:** Given a task to process considering available resources on the mobile device, the framework should determine the most energy-cost effective way of executing the processing task through its task scheduling algorithm.

3.1.1 Architecture Overview

Depicted in the figure below, is our proposed framework architecture. It is a client-server type architecture that performs the above requirements. It is composed of a cloud server that processes tasks remotely, a state machine which is the main controller of the system, a servlet container that processes the task locally and an external, physical context sensor. The state machine and servlet container are hosted on a mobile device in addition to two other components resource manager and service that handle communication to the servlet and server, and external sensory device. A pictorial description of our algorithm is given below:

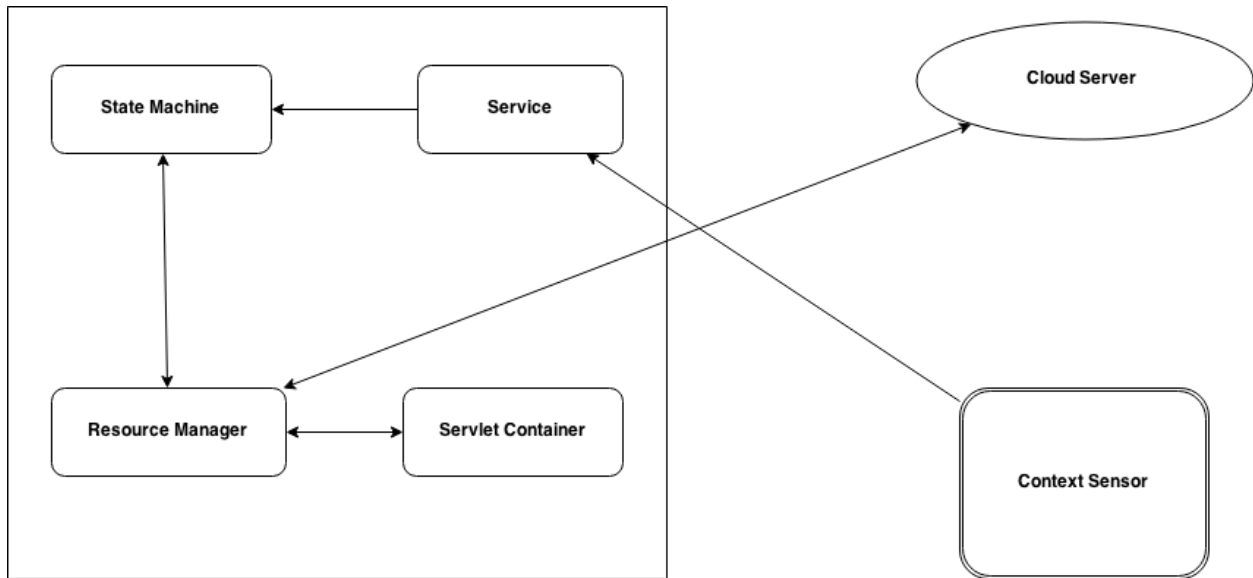


Figure 1: The System Design

The unidirectional arrows in the above diagram signify a one-way communication between components where one component sends data or messages only and the other receives only while the multi-directional arrows simply means that both components can send and receive data or messages. A summary of the roles of the major components is listed below:

- **State Machine:** We implemented the state machine on a mobile device. The state machine is the central controller and backbone of the framework. It handles most of all major communication between components in the framework and manages the process flow from the retrieval of context data to the processing and displaying of said data in an understandable format.
- **Servlet Container:** The role of the servlet container is to receive raw unprocessed data from the resource manager, process it and return it in an understandable format. However, this occurs locally on the mobile device.
- **Cloud Server:** This is the remote, resource-rich alternative to processing on the mobile device. It performed duties similar to the local servlet container; the only difference being its location and its system specifications i.e. processor size etc.
- **Resource Manager:** This is the central processing module of the framework. It fosters communication with the cloud and local service for transferring raw data and receiving

processed output. It also initiates the decision-making offload scheduling algorithm that assesses a task and determines the most suitable execution location i.e. remote or local.

- **Service:** We implemented a service on the mobile device that ran in the background and fostered communication with the external physical sensory devices. It handled the retrieval of context information from these devices. It made such data available to the state machine.
- **External Context Sensor:** Context sensors are small, embedded, resource-constrained, portable devices that can be used to measure and retrieve real-time values of a physical property within a specified environment. Examples of these are motion sensors, proximity sensors etc. We employed these to extract context information about a specific property within a given environment like humidity, temperature.

3.1.2 Context-aware Offload Scheduling Algorithm

This is one of the main functions of our framework. The algorithm is implemented within the resource manager. The scheduling algorithm assesses a processing task and determines the location for execution that would maximise the most energy conservation on the mobile device. In this process, we consider two factors the cost of local computation and the cost of communication i.e. how much mobile energy would be expended in sending the data to be processed to a remote location. The trade-offs between these two values influence the decision that is made. In other words, if local computation cost is larger than communication cost, offloading is advised and vice versa.

Our proposed task scheduling algorithm was mainly leveraged to determine the most energy-saving task execution option. Our research focused more on parsing context data and so we fine-tuned our model to determine if the task of processing said data should be done remotely or locally. In the development of this algorithmic model, we considered the following parameters:

- L_{cpu} : This denotes the current usage of the CPU of a mobile device before processing a given data.
- L_{ram} : Similar to the above, it constitutes the amount of RAM of the mobile device that is being used.

- E_t : This is the execution time of the task that is processed. In other words, how long would executing a specified software artifact take?
- I: This is the intercept of the prediction model. It can be described as the value that will be predicted for the dependent variable if the values of the independent variable was simultaneously zero.
- Regression Weights (a_0, b_0): These correspond to the values that are extracted when regression analysis is performed on our training dataset. The coefficients are estimated in order to reduce the mean squared difference between an estimated value and the actual measurement.

In achieving our goal which is energy conservation during resource-intensive processing, we view the scheduling problem as local computation cost vs. communication cost. Local computation cost refers to how much executing a given task will cost the mobile device if it is executed locally. We deduce that the execution time of a task is an effective measure of the cost of a process to the mobile system. We base our deduction on work done by (Luis Corral et al) where they demonstrated that the relationship between execution time of a process and the amount of energy (i.e. Battery life) consumed by said process is consistent. Communication costs can be defined as the costs incurred by the mobile device while transmitting data from the mobile device to a remote location. Based on the relationship between process execution time and energy consumption, computation offloading would help in conserving energy of the mobile device if the communication costs are lesser (Kumar, Liu, Lu, & Bhargava, 2013). We propose a regression-based prediction model that will produce an estimate of the execution time based on observations of the execution time when the same processing task was executed previously. Our ramifications for using this model to predict the execution time would be given in the next section.

3.1.2.1 Prediction Model

In developing the prediction model for the execution time of a temperature processing function, we considered, the L_{cpu} , L_{ram} , of the mobile device at that instance in time and the amount of temperature data, D that would be processed. We considered these metrics because after carrying out simulations of resource-intensive functions on a mobile device, we noticed that introducing a variance in the data to be processed caused a similar variance the amount of time spent waiting for a result, ergo the execution time. The same can be said for CPU and RAM loads as we noticed that when the device's processors were doing much work, the amount of time taken to execute resource-intensive functions were longer than when they were handling a lesser volume of work. In order to get the predicted execution time, we employed the use of a prediction function that we trained using multiple linear regression analysis on execution times recorded from previous instantiations or executions of the process. The resultant function is basically a linear equation. It is worthy of note that our mode of predicting the execution time is loosely based on work done by (Karl-Oskar Masing, 2013). The procedure we undertook in order to procure an optimal prediction function is given below. In our procedure, we analysed the prior execution times of a temperature processing function in correlation with our afore-mentioned scope.

3.1.2.1.1 Prediction Model Development

We proposed several possible ways of predicting the execution time of a function and we employed the use of regression analysis; specifically multiple linear regression in order to ascertain the most optimal model. The procedure is listed below:

1. We generated a data set of 12000 elements. 80% of which we used as our training data set. Each element in the data set was a tuple of $(L_{cpu}, L_{ram}, D, E_t)$ where D is the data size that constantly varied with values between 0-1000 items. Each item corresponds to a temperature value.
2. Subsequently, we implemented several execution time prediction models listed below. Due to time constraints, we considered the three most plausible equations that were logically sound to predict execution time of temperature processing function.

$$\begin{aligned}
\text{i.} \quad & \sum_{r_i \in R} \left(\frac{D}{1-L_i} \right) \cdot w_i \quad \text{where } R = \{ r_i \mid 1 \leq i \leq N \} \\
\text{ii.} \quad & \sum_{r_i \in R} (D \cdot (1 - L_i)) \cdot w_i \quad \text{where } R = \{ r_i \mid 1 \leq i \leq N \} \\
\text{iii.} \quad & \sum_{r_i \in R} \left((D \cdot (1 - L_i)) + L_i \right) \cdot w_i \quad \text{where } R = \{ r_i \mid 1 \leq i \leq N \}
\end{aligned} \tag{1}$$

In these equations, r correlates to CPU or RAM and w corresponds to the regression weights (a_0, b_0) . The weights and the intercept, I are the outputs of performing regression analysis on the training dataset. We set our prediction function as the third equation above. In subsequent steps in this procedure, we will examine the measures taken to determine this.

- Next, for each prediction model, we transformed the training data set to match them and input the transformed data set into the multiple linear regression function in R to extract the Intercept and weights i.e. I , a_0 and b_0 . This means that, given a tuple of the training data set and the prediction equation (iii), we transformed the tuple to Table 3. This was done in order to align with the regression prediction equation as depicted in section 3.1.2.2. Using the values in the tuple from the original data set in Table 2, we created the transformed dataset Table 3.

CPU Load	RAM Load	Data size	Measured Execution Time (in ns)
0.2041	0.7554	416	1622240

Table 2: Single Training data set tuple

C	M	Z
331.2985	102.509	1622240

Table 3: Transformed training dataset

We depict one of the tuples of the training and transforming data set which were transformed by calculating C as $C = (D \times (1 - L_{cpu})) + L_{cpu}$, $M = M = (D \times (1 - L_{ram})) + L_{ram}$, while $Z = E_t$. Finally, we input the transformed training dataset into the Multiple Linear Regression function in R and retrieve the regression weights and intercept.

4. Finally, we used the remaining data in the primary data set to create a validation data set with elements similar to the training dataset in order to assess the accuracy of each of the predictor equations. The regression weights are substituted into the predictor functions in order to predict the execution time of a function, given the current CPU load, RAM load and data size. We computed the Mean Squared Error (MSE) of the prediction functions over the validation data set and picked the predictor function with the lowest MSE value. Mathematically, we computed MSE as:

$$\frac{\sum_{n=1}^N (p_n - a_n)^2}{N} \quad (2)$$

where N = Size of the dataset, p = predicted execution time and a = Actual execution time. After running the procedure on all three prediction equations, we found that equation (iii) had the lowest MSE.

Our execution time prediction function originally based on the regression prediction model, $Z = a_0C + b_0M + I$ is now depicted as

$$E_t = (26090) ((D \times (1 - L_{cpu})) + L_{cpu}) + (-8536) ((D \times (1 - L_{ram})) + L_{ram}) + 308208 \quad (3)$$

3.1.2.2 Scheduling Algorithm

The equation above serves as the prediction model that we implemented in our task scheduling algorithm. Our algorithm schedules the execution of a given processing task locally or at a remote location depending on the comparison of the communication cost with the local computation cost. Our algorithm is given below:

Definition 1 (Computational Resource Usages). Let $\mathcal{R} = \{r_i | 1 \leq i \leq \mathbb{N}\}$ be a set of computational resource usages (e.g., CPU load, RAM load etc). Each $r \in \mathcal{R}$ is defined as a tuple (\mathcal{ID}, ν, w) where

- \mathcal{ID} denotes the identification of the resource (e.g., CPU, RAM etc.).
- ν represents the current load usage value of the resource.
- w represents the regression weight of the resource.

Let \mathcal{Z} be the predicted local process time computed from:

$$\mathcal{Z} = I + \sum_{i \in |\mathcal{R}|} (\mathcal{D} \cdot (1 - \nu_i) + \nu_i) \cdot w_i \quad (4)$$

where I is the intercept, and \mathcal{D} is the data size.

Let \mathcal{C} be the communication latency, which is computed by:

$$\mathcal{C} = \frac{\mathcal{D}}{\mathcal{B}} \quad (5)$$

where \mathcal{B} is the current network bandwidth.

If $\mathcal{Z} > \mathcal{C}$ then the task will be offloaded to Cloud. Otherwise, the mobile device will perform the task locally.

The above algorithm is initiated by the resource manager. Upon initiation, the algorithm computes the CPU and RAM loads of the device at that instance and inputs them into the prediction function in order to get the execution time. Subsequently, the algorithm will compute the communications cost of the device and compare it with the predicted execution time. If the execution time is less than the communication cost, the decision is sent to the resource manager that forwards the data to the appropriate location for processing.

3.2 Summary

We introduced our service-oriented framework whose aim is to maximise energy conservation of smartphones when handling resource-intensive tasks. In this chapter, we described the architecture of the framework and we also expatiated on the roles of its major components like the state machine, the resource manager, and servlet container. We also gave a brief overview of the role of the scheduling algorithm in the framework.

The state machine retrieves raw data from the service that connects to the external context sensor, extracts context information and sends it to the resource manager which assists by analysing the processing task given the size of data and infers that the handling of the task should be either offloaded or computed locally.

4 Implementation

In the preceding chapter, we introduced the design and structure of the framework which we developed to facilitate maximization of energy conservation on mobile phones whilst executing complex, recurring tasks. In this chapter, we expatiate on the development of said framework as a proof-of-concept prototype. We particularly emphasize on provisioning energy efficient continuous mobile cloud offloading through employing the task scheduling algorithm we introduced earlier; as this is the main contribution of this research study. First, we look through the various technologies, frameworks and methods that we have adopted in implementing our prototype. Then, we examine its specifics and inherent components.

4.1 Mobile Systems

This section explains the entirety of the respective mobile systems we used in our method. We would briefly examine smartphones, mobile operating systems and their role in our implementation.

4.1.1 Smartphones

The mobile device can be defined as small, handheld device which can be used for computing purposes. Such devices have in-built operating systems upon which several user applications otherwise known as apps can be launched. In this study, we focus on the Smartphone. Much like the desktops and laptops in circulation nowadays, modern smartphones can pretty much handle advanced processing.

With the advent of the Smartphone, mobile applications have become much more advanced and refined. The content on smartphones have pushed the limits of mobile application development either in the type of software technology used or the logic implemented in differing mobile applications. The rate at which smartphones are being networked and the geometrically increasing amount of data that they handle has spurred a lot of research. Advances in the design of smartphones have made it possible for users to perform at the click of a button, tasks that were previously deemed impossible or too expensive to implement. Today's Smartphone has access to the internet via Wi-Fi, 3G and 4G services, and they have many other capabilities such as sending data through NFC and Bluetooth/Bluetooth Low Energy, play music, movies and complex games, capture and manage photos and so on. Most modern smartphones have a flurry of sensors embedded in them such as temperature sensors, GPS etc. and this has introduced a new series of peculiar applications on smartphones. Few of such applications are the Google voice search which can be activated by simply saying "Okay Google!" and the proximity sensors that turn on the screen when one looks at the phone. Smartphones have been augmented to the point where they now generate, share and store multimedia data. They have the capabilities to store huge data i.e. sensor and multimedia data locally on smartphones due to improvements in their processing capabilities. There are a lot of smartphones in circulation today and this also includes the numerous, different, underlying operating systems. We would examine this in the next section.

4.1.2 Mobile Operating Systems

The proliferation of smartphones and the change in consumer tastes has influenced the development of mobile operating systems with extensive processing capabilities to handle the increasing demands. The mobile operating system is defined as a computing module that serves as a platform designed particularly for running operations and applications on a mobile device; in this context, the smartphone. There are numerous mobile operating systems; the most notable among which are Android OS, IOS, Windows Phone 8, MeeGo, and so on. Most mobile operating platforms have application stores through which user-friendly applications can be easily installed. Two prominent ones are the Apple App Store and the Google Play Store. It is worthy to note as we have stated in preceding chapters, that irrespective of the Smartphone brand or mobile

operating system, the battery life is poor and has been a constant bane to consumer utility. In our implementation, we used a Google Nexus 5 Smartphone that ran the Android v 5.0 Operating System for testing our offloading algorithm. A brief expose on the Android OS is given in the next sub-section.

4.1.2.1 Android Operating System

The Android operating system is an open source OS developed by Google and the Open Handset Alliance (OHA) and designed primarily for touch sensitive mobile devices such as Smartphone and tablet computers. The Android platform uses the Dalvik Virtual Machine to execute applications. It consists of a kernel based on the Linux kernel with middleware, libraries and APIs written in C and application software running on an application framework which includes Java-compatible libraries based on Apache Harmony. Android is more open in nature relative to other mobile platforms such as Apple's IOS in that there is less red tape when it comes to developing applications on the platform. This and more make it an acceptable platform for distributed mobile infrastructures(Nimodia & Deshmukh, 2012).

4.2 Cloud Computing

Cloud computing (Mirashe & Kalyankar, 2010) simply put is defined as the provision of computing resources or services as a utility over a given network. This is quite akin to the same way social amenities (i.e. water, electricity) is provided and paid for on a per-use basis. Technically, Cloud computing is defined as a paradigm that consists of leveraging clusters of remote servers that allow data to be uploaded for processing so as to generate results for the consumer in real time without the need for any local data storage. This paradigm is based on the following:

- **Thin clients:** These are applications that make use of the virtual computing interface.

- **Grid Computing:** This combines the resources of a network of machines in order to allow them to be used as one infrastructure.
- **Utility Computing:** As earlier mentioned, computing resources are provided to end-users based on a pay-as-you-use strategy. An example of this is the Amazon Elastic Compute Cloud on which user applications can be launched on a virtual machine in the Amazon computing environment.

Cloud systems are classified according to their mode of deployment into public, private and hybrid cloud systems. These are depicted in the figure below:

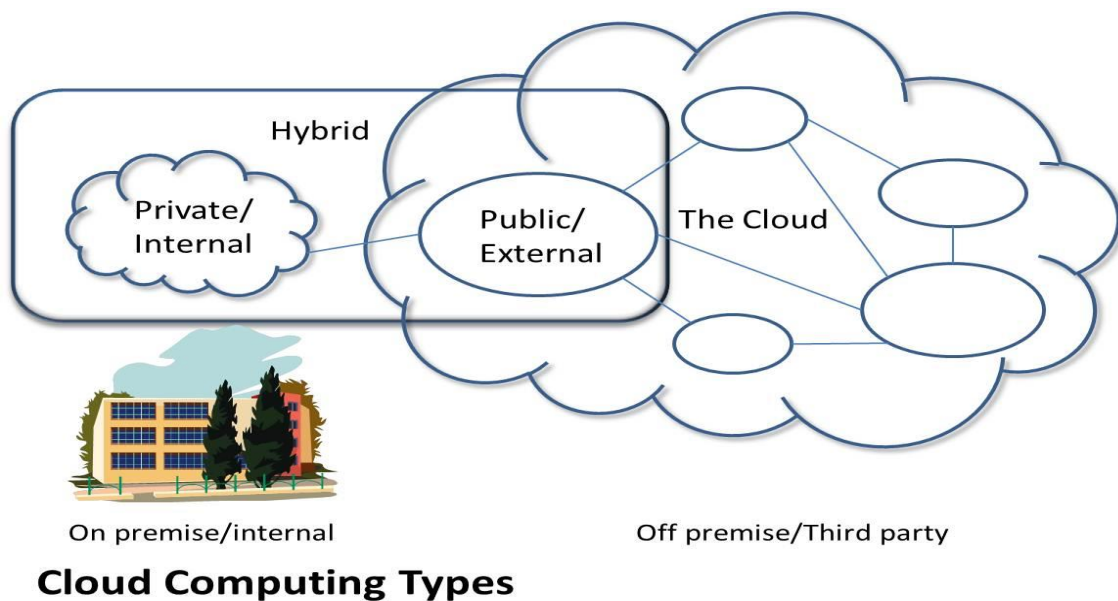


Figure 2: Cloud Computing Models¹

Each of these models are further examined below:

- **Public Cloud Systems:** This is a model that consists the involvement of a third party provider who supplies cloud services over a public, easily accessible network. In this model, the consumers pay for the amount of system resources they use. In some cases, it might be completely free. However, it has been noted that the free versions of these services

¹ <http://www.synergygs.com/Solutions/CloudServices>

are often lacking in security (18). An example of public cloud service providers are Amazon AWS, Google Compute Engine and Microsoft Azure.

- **Private Cloud Systems:** This is sometimes referred to as internal cloud (18). Private cloud systems are operated mainly for the consumption of one particular organization. Only users within that organization would have access to the custom cloud services hosted and managed on that cloud internally or by a third party firm.
- **Hybrid Cloud Systems:** The Hybrid cloud system is an amalgamation of the public and/or private cloud systems. Hybrid cloud systems leverages the benefits of either deployment model and combines them to provide a wider range of services to the user.

4.2.1 Cloud Computing Services

The services that cloud systems offer to end users can be categorized into three basic models which are examined thus:

- **Software as a Service (SaaS):** This is a model in which cloud providers provision user application software and systems over the internet. It is otherwise known as software on-demand. In some cases, they are referred to as web services. An example is Google Docs and Microsoft's Office 365. Consumers are able to access these applications from any location as long as they have internet access. Cloud providers basically handle the installation, operation and management of the application software in the cloud. They also deal with the management of the platform upon which the application runs and as such, users do not need to install the application on their local system. Most SaaS applications are accessed through the browser which simply serves as a single abstraction layer that hides all the complex processing underneath. Most versions of the application software are usually priced on the pay-as-you-use principle.

- **Platform as a Service (PaaS):** In this model, PaaS providers dispense a computing structure that serves as a platform upon which users can host their different applications. The computing platforms that host end software are usually operating systems, development environments, databases etc. This is very beneficial especially for software developers as it eliminates the need to spend resources on managing the underlying software/hardware infrastructure. An example of PaaS providers are Heroku, and Google App Engine.
- **Infrastructure as a Service (IaaS):** IaaS is a cloud service model which entails the provision of computing infrastructure be it physical or virtual machines over a network to end users. Computing infrastructure in this context pertains to an instance of a virtual server or many virtual servers, file storage, IP addresses, load balancers and other similar infrastructure. These virtual components are provided to end users to enable them, if they so desire to develop their own platforms for software deployment. Similar to PaaS, they can also serve as a platform for deploying custom scalable software depending on which infrastructure the cloud providers dispense to the application developers. An example of IaaS is Amazon EC2, Microsoft Azure and Google Compute Engine. In the next subsection, we would examine the IaaS system we leveraged in our method which was Amazon EC2.

4.2.1.1 Amazon Elastic Compute Cloud

Amazon Elastic Compute Cloud (EC2) (Amazon, 2011) is one of the web service modules within Amazon Web Services which is Amazon's comprehensive cloud platform that provides cloud services to ordinary users and developers alike. Amazon EC2 is a platform that allows users to launch and run their custom applications on its computing infrastructure. In order to access this web service, one simply has to boot an Amazon Machine Image which creates an instance which is basically a virtual machine upon which a user can launch his own software application. Users usually pay for the computing resources they utilize in running their applications mostly based on instance-hours or data transfer. The EC2 also allows users to select the geographical location or

region of their instance. We would briefly examine the types of instances that users can launch and the features of an Amazon EC2 instance.

4.2.1.1.1 Amazon Elastic Compute Cloud Instance types

An instance is basically just a virtual machine launched on Amazon EC2 which acts as a virtual server. EC2 allows users to create and launch as many instances as needed provided they pay for as many as they instantiate. Instances consist of a varying mixture CPU, memory and network capacities and each instances contains one or more instance sizes that handles the scalability of a user's resources based on what the user needs. The instance types are delineated based on the payment regime and there are three types given below:

- **Dedicated Instances:** These are instance types where the user pays per hour. It is also known as on-demand instance. This type of instance has no special discounted package or long-term customer commitment attached to it.
- **Reserved Instances:** These instances charged much less than on-demand instances as one gets a discount on the hourly charge. This comes with a long-term commitment package as the user reserves the instances for up to three years.
- **Spot Instance:** In this type of instance, the usage of an Amazon Instance goes to the highest bidder. Users bid on a spare EC2 instance and are able to run those instances when their bids exceeds the spot price. The availability of an instance for usage is quite unpredictable as the value of the Spot price is based on real-time fluctuations in demand and supply.

4.2.1.1.2 Amazon Elastic Compute Cloud Features

In this section, some of the important features (Amazon, 2011) that make using Amazon EC2 to launch software applications beneficial are examined below:

- **Auto Scaling:** Amazon EC2 allows users to scale their instances up or down according to their processing needs. It is very helpful as it can automatically regulate the amount of instances running in order to enhance performance and reduce costs. This makes the premise of scalability a lot less cumbersome.

- **Elastic Load Balancing:** This feature of Amazon EC2 is responsible for dispersing incoming traffic from a user's application across multiple cloud instances and availability zones. By so doing, it increases the fault tolerance of said application.
- **Multiple Locations:** The Amazon EC2 service is hosted in numerous locations across the world. These locations comprise of Regions which are a designated geographical area decomposed into several Availability zones. Amazon EC2 provides users with the ability to create instances and replicate them across multiple locations or availability zones and this helps preserve the availability of an instance in the rare event of a system failure. This is because availability zones have been developed to resist failures in other zones.
- **Amazon Elastic Block Store:** The Amazon Elastic Block Store (EBS) is used to provide persistent storage for Amazon EC2 instances. It provides persistence independent of the operation flow an instance. The EBS volumes are network-connected, reliable and available volumes that allow users to create snapshots of the state of a user's volumes which are then replicated across all availability zones (Amazon, 2011).

4.3 Service-Oriented Architecture

A service-oriented architecture is a paradigm that involves developing applications as loosely coupled, well defined, modules that perform specific actions and exist independent of each other, but can be coordinated to perform a particular function. These modules are services. A service is a stand-alone, self-contained unit of functionality that performs at least one action within a specific information system and acts independent of the state of other services. It is not necessary for a service to know the computation details of the other services in the system in order to interact with it. The interaction between one or more services is dependent on the specification of the application that leverages them. We employ this pattern in developing our framework and prototype as it facilitates platform-independence and code re-use (Schroth & Janner, 2007).

4.4 Arduino

Arduino (Wikipedia, 2012) is a lightweight, inexpensive open-source physical computing platform used mostly in building digital or electronic devices that perform an array of simple and complex duties such as sensing temperature and humidity data. They can either be stand-alone, singular implementations or be programmed to communicate with external devices. An example of real world Arduino implementations are thermostats, motion detectors and robots. We depict in this section, a description of the Arduino Board, the Arduino IDE, Arduino Shields and Sensors that we used in our implementation.

4.4.1 Arduino Board

There are different kinds of boards with varying specifications and functionalities. The most popular among which are Arduino Uno, Arduino Leonardo, Arduino Mega ADK and a host of several others. The standard Arduino board consists of an Atmel 8-bit, 16-bit and 32-bit AVR microcontroller, a USB port which can be used to power the board through connecting to a desktop or laptop, a power outlet which can be used to power the board through the AC plug, 14 digital input/output pins and 6 analog input pins. These pins are mostly used to extend the functionality of the Arduino board by connecting Shields (i.e. Bluetooth, Ethernet, Wireless or GPS Shields) and/or breadboards to them. These boards can be assembled by hand or purchased already assembled while the Arduino Development Environment (Arduino DE) can be downloaded for free. In this paper, we made use of the Arduino Mega ADK as an electronic platform upon which we connected our temperature sensor. More details will be given in the next chapter. Depicted in the figure below is an Arduino Mega ADK board:



Figure 3: Arduino Mega ADK Board²

4.4.2 Arduino Development Environment

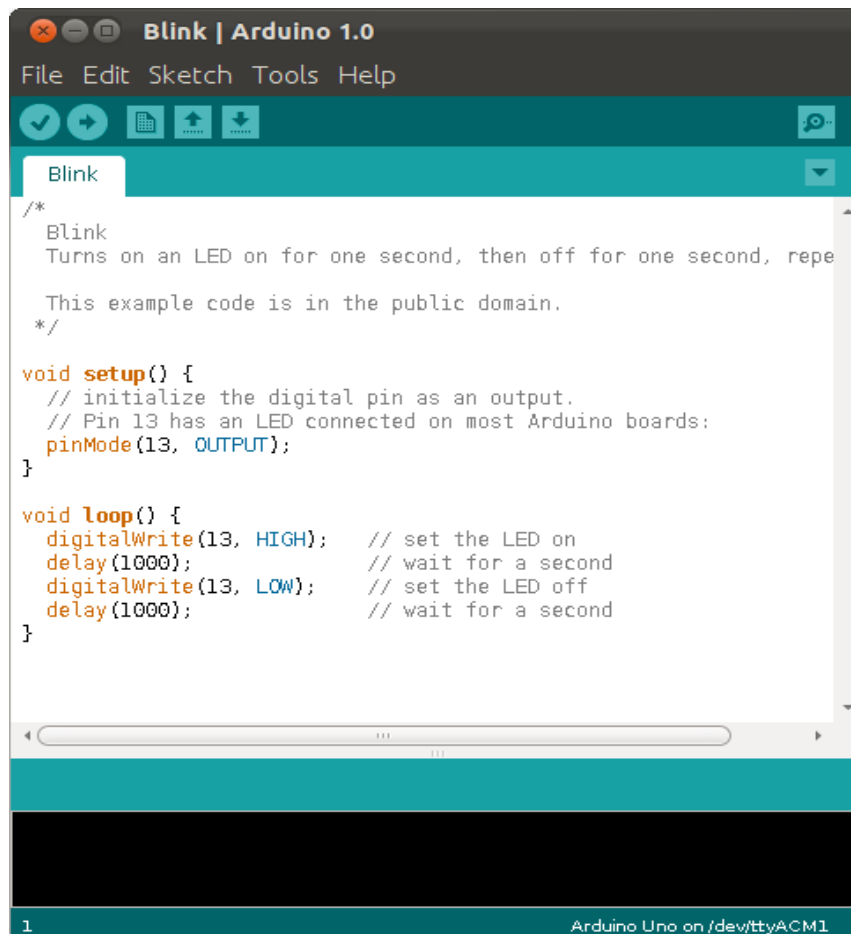
The Arduino DE (Arduino Development Environment, 2015) is an open-source, platform-independent, software written in Java that can be used to programme Arduino boards. The Arduino programming language used to program its boards is based on Wiring (Wiring, 2015); another physical computing environment and Processing programming language (Processing, 2015). It contains a text editor where code is written, a toolbar containing buttons with varying functions, a text console, a message area and various menus. In the text editor, text written can be cut/pasted and commented out. It has the ability to compile and upload code written in the editor, known as Sketches in Arduino to the connected Arduino board. It can also display any possible error or information while uploading to the board.

4.4.2.1 Arduino Sketches

Arduino Sketches are the programs that are written in the Arduino software, compile and uploaded to a connected Arduino board. The sketches are written in the text editor and are saved on compilation with an .ino file extension. Arduino sketches are written mostly in C or C++. The Arduino DE comes with an in-built Wiring library that makes input and output operations less cumbersome. In order to effectively run and execute a sketch, two functions are imperative; namely

² <http://www.arduino.cc/en/Main/ArduinoBoardMegaADK>.

setup and loop. The setup function executes constant Arduino parameters only once while whatever code is within the loop function is executed concurrently depending on the amount of delay that is introduced. To better understand this, a pictorial description of the Arduino DE with a sketch that turns an LED light on and off is given below:



```
/*
 * Blink
 * Turns on an LED on for one second, then off for one second, repe
 *
 * This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH); // set the LED on
  delay(1000); // wait for a second
  digitalWrite(13, LOW); // set the LED off
  delay(1000); // wait for a second
}
```

Figure 4: Arduino DE with a Sketch³

4.4.3 Shields

Earlier on in this chapter, we mentioned that a basic Arduino board has a series of pins to which devices known as shields can attached to the board. The purpose of these shields are to extend the function of the Arduino board. That is to say, the shields increase the capabilities of the board.

³ <http://en.wikipedia.org/wiki/Arduino>

Within Arduino, there are different kinds of shields namely Arduino Ethernet Shield, Arduino Wi-Fi Shield etc. However, within the context of this research study, we did not employ the use of an Arduino Shield, rather we used a RedBearLab BLE Shield. We would explain this further in this chapter. In this section, we would briefly examine the shields we used in our implementation.

4.4.3.1 RedBearLab BLE Shield

The RedBearLab BLE Shield is a Bluetooth Low Energy Shield designed to work with Arduino boards. It is statically a BLE peripheral device used to connect to BLE central devices like a Smartphone. Since it runs between 3.3 to 5v, it can also be used with other Arduino-compatible boards to facilitate custom applications. It contains flexible RDYN and REQN pins that can be placed between pins 2 to 12 but are fixed at pins 8 and 9 respectively on the Shield. It also has an LED light that indicates whether the Shield is properly connected or not and a reset button which also resets the Arduino board. Given in the figure below is a depiction of the RedBearLab Shield:



Figure 5: RedBearLab BLE Shield⁴

⁴ <http://redbearlab.com/blshield>

4.4.3.2 TinkerKit Mega Sensor Shield

The Tinkerkit Mega Sensor Shield serves as a platform upon which sensors can be attached to the Arduino board specifically the Arduino Mega ADK. The sensor retrieve real-world, physical data and transmit it to Arduino. It consists of 22 standard TinkerKit 3 pin connectors, 10 analog inputs and 6 analog outputs connected to the Arduino board's output pins. It also has an LED light that indicates whether the Shield is properly connected or not.

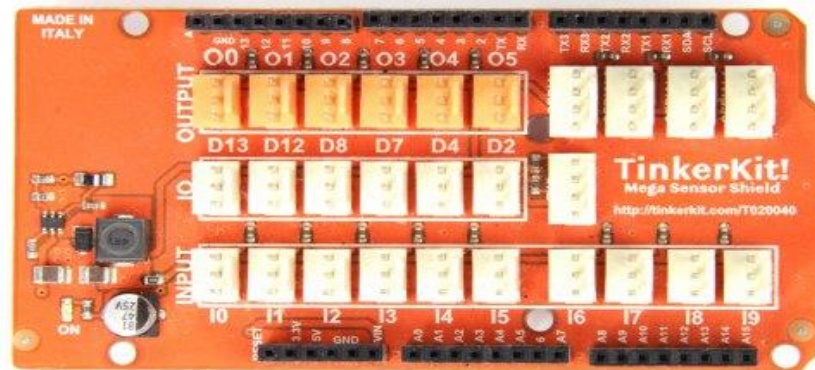


Figure 6: TinkerKit Mega Sensor Shield⁵

4.4.4 Sensors

In this section, we examine the sensors we employed in our prototype.

- **TinkerKit Thermistor Module:** The thermistor is simply a resistor that varies its resistance inversely proportional to the perceived temperature. In other words, it decreases its resistance when the temperature of the environment to be measured increases, and vice versa. It outputs analog values between 0 and 1023 when it is connected to an Arduino board through the TinkerKit Mega Sensor Shield. The TinkerKit thermistor Module consists of a thermistor, the standard TinkerKit 3pin connector, a signal amplifier, green

⁵ <http://store.arduino.cc/product/T020040>

and yellow LED lights which indicates the sensor's connected state and the variation in temperature respectively.

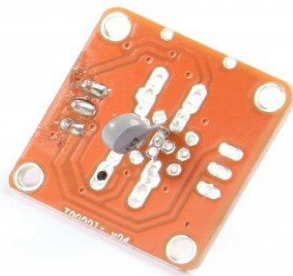


Figure 7: TinkerKit Thermistor Module⁶

In our prototype, since our focus was mainly on processing continuously provisioned context information, we used an amalgam of an Arduino micro-controller, a RedBearLab BLE shield for communicating with the Service and a TinkerKit Mega Sensor Shield with a thermistor module attached. The sensor retrieves context information such as the temperature of a place in real-time. This is depicted in the picture below:

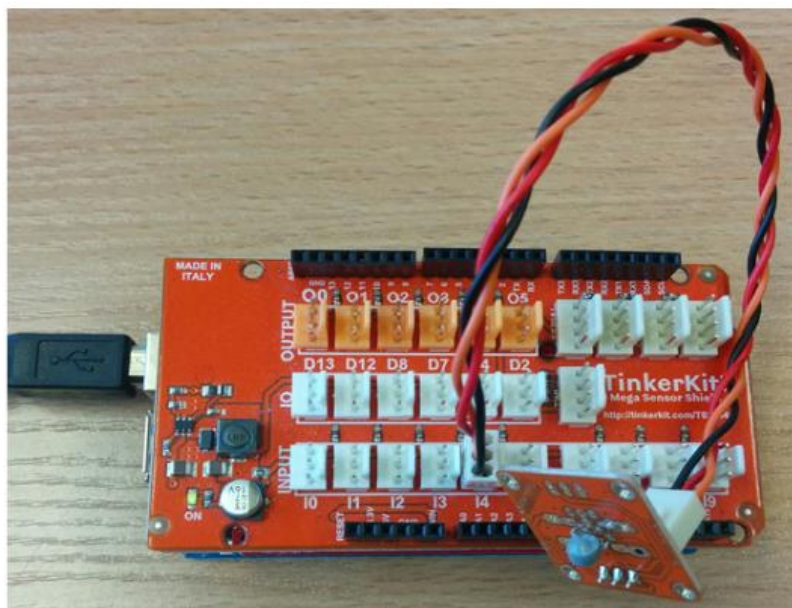


Figure 8: Temperature Sensor Node

⁶ <http://store.arduino.cc/product/T000200>

4.5 Machine Learning

Machine learning (Mitchell, 2006) is simply a way to get machines or systems to act and perform tasks autonomously without any prior explicit programming. Machine learning systems leverage algorithms or programs that learn from and make predictions about certain concepts based on given data. These algorithms are usually trained on real-time observed data of a specified concept in order to actively make predictions on new unknown data culled from a different instance of the same concept. The aim of this, is to improve the program's understanding and behaviour in dealing with different instances of the same observed concept. Although similar to data mining systems, which identifies recurring patterns in data and extracts said data for further processing, a machine learning system learns from recurring patterns in data and modifies itself accordingly. The concept of machine learning has been used in many real-world applications, some of which are self-driving cars, social network analysis, spam identification and filtering, and in the field of augmented reality. Machine learning has been implemented in different algorithms such as regression, support vector machine, decision trees etc. Machine learning approaches are categorized into three main categories namely: supervised learning, unsupervised learning and reinforcement learning. A minor category is semi supervised learning. We briefly expatiate on these in subsequent subsections.

4.5.1 Supervised Learning

This is a machine learning approach that involves learning and predicting new behaviour using a given training dataset about which prior knowledge exists. By prior knowledge, we mean that the type, categories, classes or the labels of data are specified and the machine learning algorithm makes predictions based on the input i.e. training data. Another way to look at this concept, is that supervised learning predicts the output of a system based on series of inputs (Love, 2002). There are many algorithms that implement supervised learning such as Support Vector Machines (SVMs), Linear Regression and the Naïve Bayes algorithms. In this study, we implemented a supervised learning algorithm in order to clearly define the prediction model we used in our method. We peruse this in the next sub-section.

4.5.1.1 Linear Regression

Linear Regression (Su, Yan, & Tsai, 2012) belongs to the class of regression analysis algorithms. Regression analysis is a process concerned with predicting or forecasting the relationships between variables. It employs many techniques to identify, given a dataset, which variables are dependent on other variables i.e. independent variable. It also infers what the nature of the relationship between the dependent and independent variables entail. In other words, it answers the question of how the value of the dependent variable varies with fluctuations in the value of the independent variable. Regression analysis has many forms and one of such is linear regression. Linear regression has to do with modelling the interaction of a singular dependent variable with one or more dependent variable. Given a training dataset of independent and dependent variables, linear regression can be used to fit a predictive model to the data in order to train the model to estimate the value of a dependent variable in a dataset where the independent variable is given. In this study, we employed the use of multiple linear regression (Fletcher, 2009). Multiple linear regression is an extension of simple linear regression (Brandimarte, 2011) and in this form, more than one independent variable is used to predict the value of a single dependent variable. Mathematically, it can be expressed as such:

$$Y = b_0 + b_1X_1 + b_2X_2^7 \tag{6}$$

In this equation, two independent variables i.e. X , predict the value of a single dependent variable Y . The “ b ” values are the regression weights which are present in simple linear regression. They are determined in such a fashion that they minimize the Mean Squared Error.

⁷ <http://www.psychstat.missouristate.edu/multibook/mlt06m.html>

4.5.2 Unsupervised Learning

This approach (Ghahramani, 2004) is the direct opposite of supervised learning. In this approach, an algorithm makes learns from and makes predictions on a given data without prior knowledge of the structure of the data. The algorithm has to identify the hidden structure of data that has no specified categorization, labelling or classification. One advantage this has over supervised learning is that one can learn and predict larger and more complex models because its prediction model is not dependent on the input values unlike supervised learning. In cases, where some input values are missing, supervised learning cannot make any predictions about the output while this poses no issues in unsupervised learning as missing input values can be inferred from other observed values in the input data set.

4.5.3 Reinforcement Learning

This approach is rather different from the previous two i.e. supervised and unsupervised learning. Reinforcement learning (Sutton & Barto, 2012) is concerned with the manner in which an artefact i.e. software program behaves within a dynamic environment in which it must achieve a certain task but most autonomously ascertain if it has achieved that task. An example of this is understanding how to play a certain game by playing against an opponent.

4.6 State Machine

We implemented the state machine using Stately.js (Stately, 2015) is a JavaScript based finite-state machine workflow engine originally developed for Node.js and the browser. However, in our implementation, we leveraged this lightweight state machine as the platform that directed the operation flow of the offloading scheme. We used this in order to demonstrate our energy conservative scheduling algorithm in a basic offloading scenario that involved retrieving temperature data from our sensor and processing it. We employed this in our test bed which was an Android Nexus 5 Smartphone and connected said phone to the Arduino board. Given below is a code snippet depicting a sample Stately.js state machine:

```

var door = Stately.machine({
  'OPEN': {
    'close': /* => */ 'CLOSED'
  },
  'CLOSED': {
    'open': /* => */ 'OPEN',
    'lock': /* => */ 'LOCKED'
  },
  'LOCKED': {
    'unlock': /* => */ 'CLOSED',
    'break': /* => */ 'BROKEN'
  },
  'BROKEN': {
    'fix': function () {
      this.fixed = (this.fixed === undefined ? 1 : ++this.fixed);
      return this.fixed < 3 ? this.OPEN : this.BROKEN;
    }
  }
});

```

Figure 9: Stately.js Sample State Machine⁸

4.7 Servlet Container

We examined the servlet container component and its role in our framework in the preceding chapter. For our proof of concept prototype we leveraged the use of I-jetty (28). I-jetty is the mobile implementation of the free, open-source, servlet engine and http web server known as Jetty. I-jetty was developed mainly for Android devices as it is only available on the Google Play Store. The Jetty project (Dirksen, 2012) is currently hosted at the Eclipse foundation. There are a lot of organizations that currently use this webserver in their products like as Google App Engine, Apache Maven, Hadoop and Eclipse. In our method, we used the mobile implementation of Jetty, I-jetty as a servlet container to handle the processing of temperature data locally.

⁸ <https://github.com/fschaefer/Stately.js>

4.8 Implementation Overview

Our method description is divided into three major phases listed below:

- Context sensor setup
- Implementing the Scheduling algorithm
- Configuring the prototype

We examine the various processes involved in completing each phase and how they all tied in to achieve our goal of mobile device energy conservation during intensive processing. In the previous chapter, we denoted that our framework is focused on parsing and processing context information from external context sensors. However, in our prototype, we focused specifically on parsing data from a temperature sensor that we have depicted earlier in this chapter. We depict an overview of the scenario in the figure below:

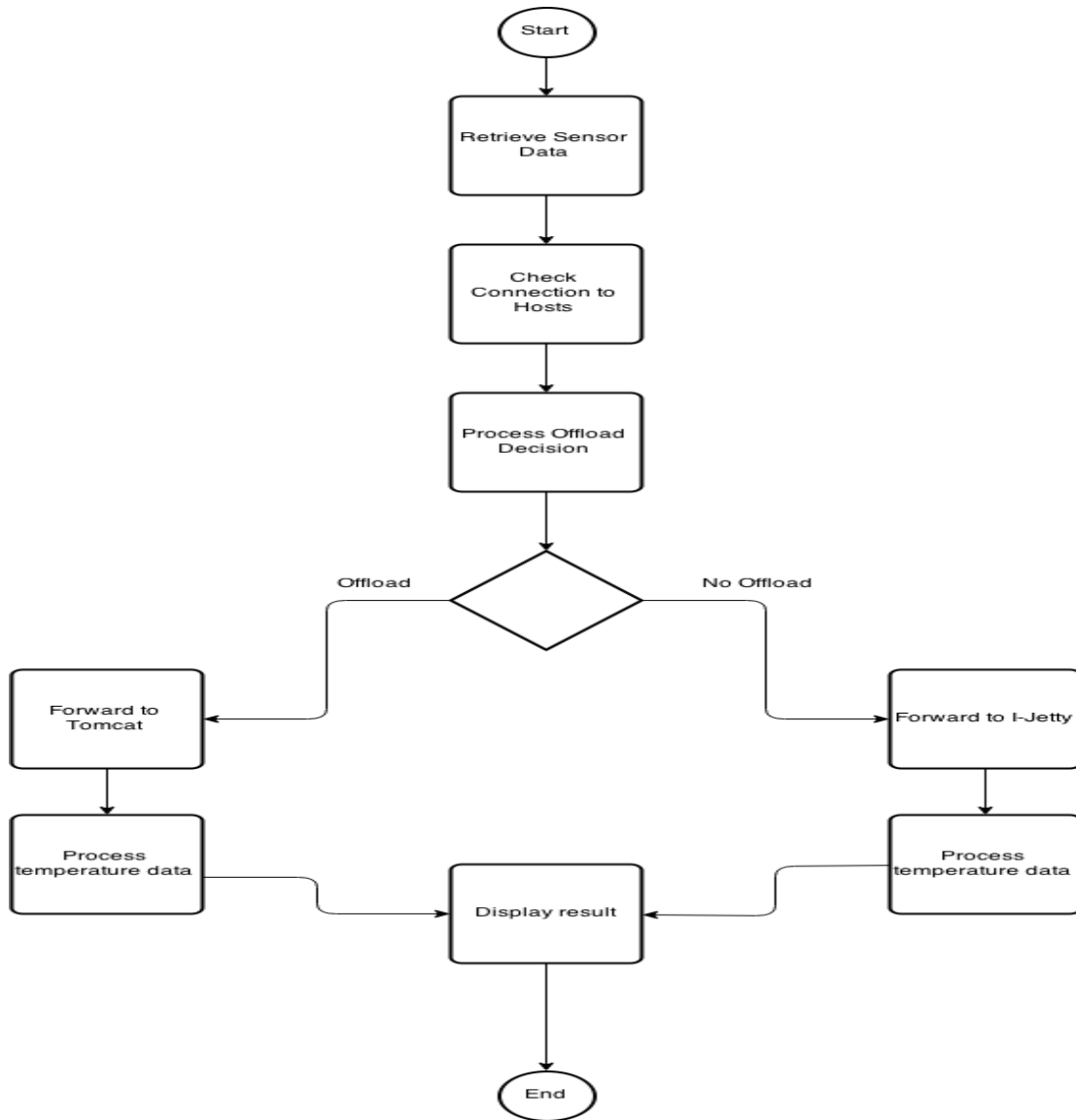


Figure 10: Temperature Parsing Scenario

We employed the use of several software and IDEs in setting up our prototype namely:

- Android Studio:** This was the major platform upon which 90% of the framework was hosted; except for the physical sensing device. Android Studio (Android Studio, 2015) is now the official IDE for Android application development. Previously it was Eclipse IDE. The Android Studio is based on IntelliJ IDEA and it incorporates a flexible Gradle-based build system converse to Eclipse that employs Maven and/or Ant which like Gradle are build automation tools.

- **Arduino DE:** We already gave a thorough explanation of the Arduino DE and its components in the section 4.3 of Chapter 4. We used the Arduino DE to programme the physical device. Further detailing will be done subsequently.
- **RStudio:** RStudio (RStudio, 2012) is an open-source development environment written in C++ for the R language which is basically a statistical computing programming language employed in many fields such as Data mining and Machine learning. We employed it in performing multiple linear regression analysis over our training data set in order to determine the most optimal prediction function. We would further discuss this in the next section.

We examine the procedures followed in using these software applications to develop the prototype and applying with respect to continuous temperature sensor data parsing. In subsequent sections, we examine all the phases of our implementation in detail.

4.8.1 Context Sensor Setup

Earlier in this chapter where we defined the technologies we employed, we talked about the open-source microcontroller Arduino. In this thesis, for our proof of concept prototype, we focused mainly on modelling the temperature data processing scenario from one external context node. In section 4.4.3, we depicted the setup of the physical sensor node which we used to retrieve data about the temperature within a specified environment. The sensor node consisted of an Arduino Mega ADK micro-controller board, a RedBearLab BLE Shield, and a TinkerKit Mega Sensor Shield with a TinkerKit thermistor module attached. The setup was powered through a USB connection to a laptop from the Arduino board. The Arduino board acted as the foundation for the custom physical node. We programmed the board to retrieve data from the thermistor module in real-time and send it to a connected Android mobile device over the BLE protocol. The connection is handled by a background service on the mobile device. This would be discussed further in later sections of this text. Given below is the code snippet depicting the process of setting up Arduino to retrieve temperature information.

```

String str;
int outPin = 4; // LED connected to digital pin 13
int inPin = 3;  // pushbutton connected to digital pin 7
int val = 0;
void setup()
{
    pinMode(outPin, OUTPUT);    // sets the digital pin 13 as output
    pinMode(inPin, INPUT);
    ble_begin();
    // Enable serial debug
    Serial.begin(9600);
}

unsigned char buf[16] = {0};
unsigned char len = 0;

void loop()
{
    if ( ble_connected() )
    {
        int value = analogRead(A4);
        Serial.println(value);
        ble_write(value);
    }
    ble_do_events();
    delay(5000);
}

```

Figure 11: Extracting sensor data Arduino code snippet

The most important section to note in the above snippet is the loop() function. Earlier in this Chapter, we talked about the function of the setup() and loop() as initializing values only once on startup of the program and recurrent execution on an interval specified by the delay, respectively. In the loop() function, when a BLE client; in this case an Android device connects to the node which is the BLE Server, the value of the temperature is read from the pin where the sensor is connected and sent back to the device. This process of data transmission is repeated every 5 seconds.

4.8.2 Configuring the Prototype

In this section, we examine the procedures involved in developing and setting up our service-oriented framework. The prototype is mainly based on Stately.js, a JavaScript state machine. We implemented this and other components such as the service and resource manager on an Android web app. We describe the setup of the prototype within the context of continuously processing temperature data from an external sensor over a specified time interval. We display the prototype architecture below.

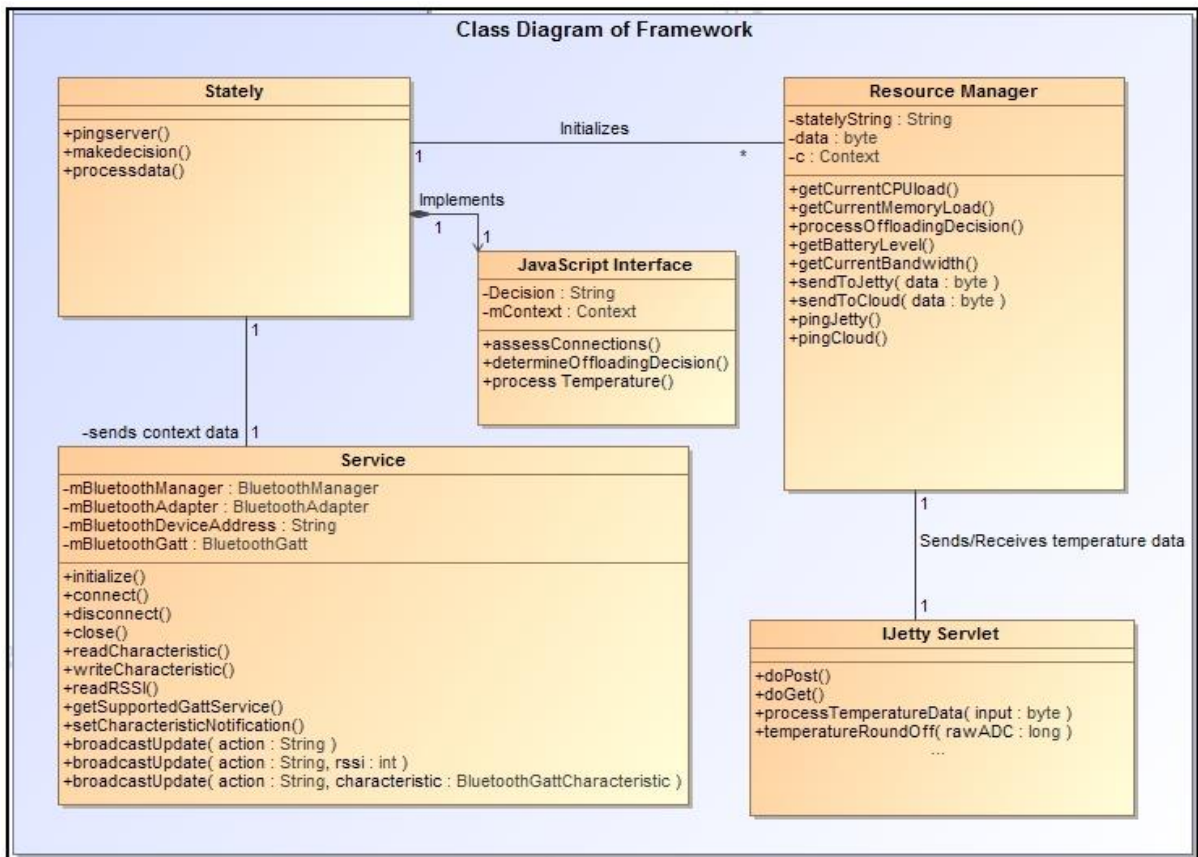


Figure 12: Class diagram of prototype

In the diagram above, we depict the major components of the system and their interactions with one another. We show how Stately.js contains methods within states to confirm if the I-Jetty Servlet and Tomcat cloud server are active, implement the algorithm in making offload decisions, and processes the retrieved temperature data. Stately.js contains a JavaScript interface which handles the translation of such directives within the android framework. We depict that Stately.js retrieves context data from the Service that connects to the external Physical device; in our case, an Arduino sensor node. We also show that for executing each of those functions it initializes the resource manager that does the actual handling. Finally, we depict the interaction of the I-Jetty Servlet with the resource manager in that it receives data, processes it and returns an understandable output. We explain in subsequent subsections, the actual procedures implemented in setting up the prototype.

4.8.2.1 Android Service Setup

The Android Service (Android, 2015) is a component of an android application which can be used to execute long-running operations in the background. This is done mostly so that the UI thread does not have to handle bulky processing on its main thread. Services are usually instantiated in two ways; they can either be started by an android application component or an application component can bind itself to an already started service. The latter was the case in our approach. We created a service to scan, and connect with an external, physical sensor node and we bound our application to the service to retrieve context information from it. In order to retrieve context information from the Arduino sensor node, we employed the use of android-specific BLE APIs.

In scanning for BLE devices, we put the permissions for Bluetooth and BLE in our manifest.xml file and instantiated the BluetoothAdapter and BluetoothGatt classes (Android Bluetooth Low Energy API, 2015). The BluetoothAdapter object corresponds to the device's own Bluetooth adapter. For the entire android mobile system, there is only one adapter and using an instantiation of this class in the service, one can interact with it in order to make use of the device's Bluetooth radio to connect to BLE devices. In order to start scanning, one must call the startLeScan () method which returns a BluetoothAdapter.LeScanCallback as a parameter. This callback function is responsible for delivering the scan results to the service. The figure below denotes the function for scanning.

```

public class DeviceScanActivity extends ListActivity {

    private BluetoothAdapter mBluetoothAdapter;
    private boolean mScanning;
    private Handler mHandler;

    // Stops scanning after 10 seconds.
    private static final long SCAN_PERIOD = 10000;
    ...
    private void scanLeDevice(final boolean enable) {
        if (enable) {
            // Stops scanning after a pre-defined scan period.
            mHandler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    mScanning = false;
                    mBluetoothAdapter.stopLeScan(mLeScanCallback);
                }
            }, SCAN_PERIOD);

            mScanning = true;
            mBluetoothAdapter.startLeScan(mLeScanCallback);
        } else {
            mScanning = false;
            mBluetoothAdapter.stopLeScan(mLeScanCallback);
        }
        ...
    }
}

```

Figure 13: Bluetooth Scanning⁹

In order to retrieve data from a BLE device, one needs to connect with the GATT server (Android Bluetooth Low Energy API, 2015) hosted on the device. In BLE protocol terminology, the BLE device is referred to as the GATT server while the android app that seeks to connect is the GATT client. We use an instantiation of the BluetoothGatt class to connect to a GATT server. Subsequently, we use the BluetoothGatt instance to conduct GATT client operations such as extracting BLE device characteristics and interacting with its given services. When an event occurs at the BLE device; for instance, new Bluetooth services are discovered or new data is available from a characteristic read, it calls a broadcastUpdate() function and passes it an action. This function is responsible for sending data and other related information from the Service to the binding application component. We display a sample broadcastUpdate() function that retrieves data from a sensor in the figure below:

⁹ <http://developer.android.com/guide/topics/connectivity/bluetooth-le.html>

```

private void broadcastUpdate(final String action,
                            final BluetoothGattCharacteristic characteristic) {
    final Intent intent = new Intent(action);

    // This is special handling for the Heart Rate Measurement profile. Data
    // parsing is carried out as per profile specifications.
    if (UUID_HEART_RATE_MEASUREMENT.equals(characteristic.getUuid())) {
        int flag = characteristic.getProperties();
        int format = -1;
        if ((flag & 0x01) != 0) {
            format = BluetoothGattCharacteristic.FORMAT_UINT16;
            Log.d(TAG, "Heart rate format UINT16.");
        } else {
            format = BluetoothGattCharacteristic.FORMAT_UINT8;
            Log.d(TAG, "Heart rate format UINT8.");
        }
        final int heartRate = characteristic.getIntValue(format, 1);
        Log.d(TAG, String.format("Received heart rate: %d", heartRate));
        intent.putExtra(EXTRA_DATA, String.valueOf(heartRate));
    } else {
        // For all other profiles, writes the data formatted in HEX.
        final byte[] data = characteristic.getValue();
        if (data != null && data.length > 0) {
            final StringBuilder stringBuilder = new StringBuilder(data.length);
            for(byte byteChar : data)
                stringBuilder.append(String.format("%02X ", byteChar));
            intent.putExtra(EXTRA_DATA, new String(data) + "\n" +
                stringBuilder.toString());
        }
    }
    sendBroadcast(intent);
}

```

Figure 14: BroadcastUpdate function¹⁰

4.8.2.2 Implementing the State Machine

Stately.js was used as the state machine in our prototype. It served as the main initiator for the framework. It started the process of retrieving temperature data from the service and sending it to the resource manager where the data to be processed is assessed and an energy-optimal execution decision to offload or compute locally is made using the task scheduling algorithm. We created an android web app upon which we hosted Stately.js. A web app is a browser-like structure hosted within an android Web View on a mobile device that allows you to display web pages and content. Since Stately.js is written in JavaScript, we created a JavaScript Interface class which acted as the bridge between Stately's browser components and Android's native methods. Given in the figure below is our web app showcasing the state machine:

¹⁰ <http://developer.android.com/guide/topics/connectivity/bluetooth-le.html>

```

<head>
</head>
<body>
<form name="test">
  <input type="text" id="nameText" value="Start Process" />
  <input type="button" value="Click me!" onclick="offloadmachine.pingserver(nameText.value).makedecision().processdata()" />
</form>
<script type="text/javascript" src="https://rawgit.com/fschaefer/Stately.js/master/Stately.js"></script>
<script type="text/javascript">
  var offloadmachine = Stately.machine({
    'START': {
      'pingserver': function(forms) {
        AndroidConnect.pingHost(forms);

        return this.PROCESS;
      }
    },
    'PROCESS': {
      'makedecision': function() {
        AndroidConnect.offloadingDecision();

        return this.EXECUTE;
      }
    },
    'EXECUTE': {
      'processdata': function() {
        AndroidConnect.processTemperature();

        return this.START;
      }
    }
  });
</script></body></html>

```

Figure 15: State Machine of Prototype

AndroidConnect is the JavaScript interface that connects Stately.js to the rest of the components. In our web app, the state machine retrieves context data sent to it from the background service, and forwards the data to the resource manager. It returns the offloading decision after employing the use of the task scheduling algorithm to determine the most energy-optimal execution alternative for the mobile device. Based on that decision, it then directs the resource manager to send the data to the I-jetty servlet container for local processing or to the Tomcat Instance on the cloud.

4.8.2.3 Implementing the Scheduling Algorithm

In this section, we present how we used our task scheduling algorithm in our prototype. Stately.js determines the most energy-optimal execution of the temperature parsing task by sending a directive to the resource manager to execute the algorithm. The algorithm takes the size of temperature data to be parsed and predicts the execution time of the process. It subsequently

computes the cost of data transfer and outputs a decision based on the comparison of these two values. This is depicted in the source code below:

```
public String processOffloadDecision(){
    String Decision = "";
    //Compute CPU load
    String cpuLoad = new DecimalFormat("#.####").format(getCurrentCpuLoad());
    double cpuValue = Double.parseDouble(cpuLoad);

    //compute RAM load
    String memLoad = new DecimalFormat("#.####").format(getCurrentMemoryLoad());
    double memValue = Double.parseDouble(memLoad);

    //get predicted time in nanoseconds
    double predictedTime = (26090 * ((this.data.length * (1 - cpuValue)) + cpuValue))
        + ((-8536) * ((this.data.length * (1 - memValue)) + memValue)) + (308208);

    //get current Bandwidth
    long Bandwidth = getCurrentBandwidth();

    //determine communication cost
    double commCost = (double) this.data.length / Bandwidth;
    //convert cost to nanoseconds
    double communicationCost = commCost * 1000000000;

    //compare prediction time with communication cost
    if (predictedTime > communicationCost){
        Decision = "Offload";
    } else {
        Decision = "No Offload";
    }

    return Decision;
}
```

Figure 16: Task Scheduling Algorithm Source Code

4.8.2.4 Servlet Container Configuration

In our approach, we presented a scalable approach to efficiently processing temperature data. We created a maven web app which contained servlets to handle inquisitions from the framework, process any given data and return it in a clear, understandable format. Compiled the servlet application into a .war file and saved it on our mobile device. We used I-jetty to handle local

processing of temperature data. I-Jetty originally was intended to be used as a web server. However, in order to fit it into our service-oriented model, we made custom modifications to the I-jetty source code to dynamically download and install our .war file from the mobile device when I-jetty was started. The .war file we generated when we developed the servlet application was also uploaded and launched on the Tomcat server in Amazon EC2 in order to also field and handle requests from the prototype.

For further understanding, we present a use-case diagram depicting the interactions of the user and other external systems with our framework such as the Tomcat Server and the sensor. We modelled these as actors in the diagram.

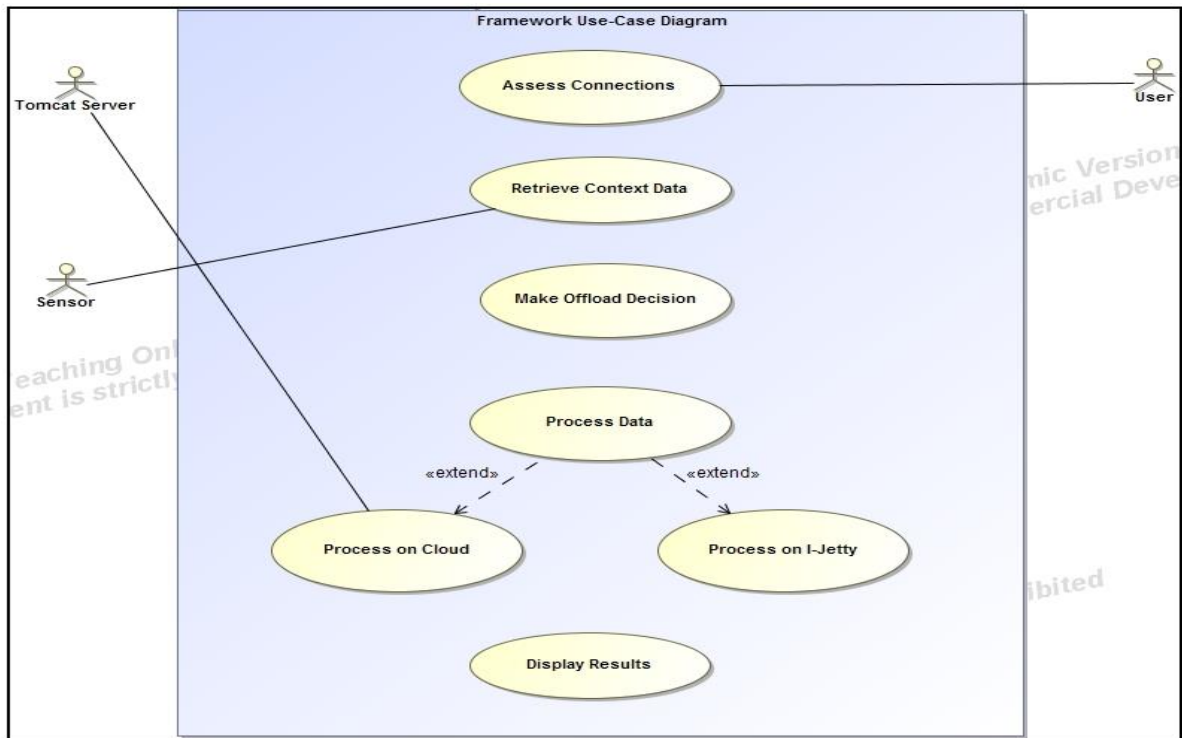


Figure 17: Prototype Use Case Diagram

4.9 Summary

This chapter presented an overview of our Implementation. We thoroughly examined the frameworks, and technologies used in implementing our solution. We examined each of the phases involved in developing our proof-of-concept prototype. We look at methods involved in sensing

and extracting context information from its immediate surroundings using the Arduino sensor node. We analyzed our task scheduling algorithm and depicted how we implemented it within our prototype. We presented a comprehensive description of our prototype and its components and reviewed their implementation.

5 Experimentation

In this chapter, we present an evaluation of our framework in order to assess its ability to conserve the energy of a given smartphone through energy-optimal task execution scheduling during continuous heavy processing. We discuss, in subsequent sections within this chapter, the case studies or test cases we implemented in order to fully assess the performance of our framework with regard to resolving the problems disclosed in Chapter 1.

5.1 Use Case and Setting

5.1.1 Energy-Conservation Evaluation

The scenario we employed for evaluating if the framework actually conserved energy was, as we have mentioned in earlier chapters, within the context of processing analog temperature values we retrieved from the external Arduino sensory node. Our measurements were taken from the instance where we sent data to the remote or local services for processing, to the time when we received a response from the services.. We executed simulations using an Android-based Google Nexus 5 Smartphone as our test bed. We considered CPU usage as a salient metric that correlates to the energy consumption of the mobile device due to the fact that the amount of processing done on the CPU has a much more effect on the battery life than any other measure.

For our testing environment, we used an augmented version of our prototype. In order to get cognizant readings, we iterated the process of either offloading or local computation multiple times. This was because we observed that a single iteration of processing values had little or no measurable effect on the CPU. So, we computed multiple iterations of the process. During each iteration, we computed the difference in the CPU loads before and after processing. We validate our framework by considering and comparing the CPU usages of offloading versus local computation. In other words, when the algorithm returns “Offload” as its decision, we process the

data on the cloud and on the local servlet container whilst computing their respective CPU usage. We depict our framework as successful if it suggests offload and the CPU consumption for processing remotely is lower than processing locally.

5.2 Results

5.2.1 Test case 1

In this test case, we evaluate and compare the CPU consumption rate of offloading the processing of temperature data to the Tomcat Instance on Amazon EC2, against computing the process locally. We consider and compare the fraction of the CPU that is consumed or used up when dealing with varying data sizes.

We denote in the figure table below our findings that correlate with the amount of CPU used in offloading.

CPU Consumption	Data size (in Bytes)
0.0112	10
0.0347	20
0.00297	30
0.010	40
0.00386	50

Table 4: CPU Consumption for offloading

We mentioned earlier in this chapter, that we aim to compare the CPU consumption rate of offloading against local consumption when the algorithm proposes an offload in order to assess its ability to decipher and proffer the most energy-optimal execution solution given the current state of the system. We present in the table below the CPU consumption rate of computing locally when the algorithm favoured offloading.

CPU Consumption	Data size (in Bytes)
0.0197	10
0.0521	20
0.0147	30
0.0051	40
0.00430	50

Table 5: CPU consumption for local computation

We present these values in the graph given below:

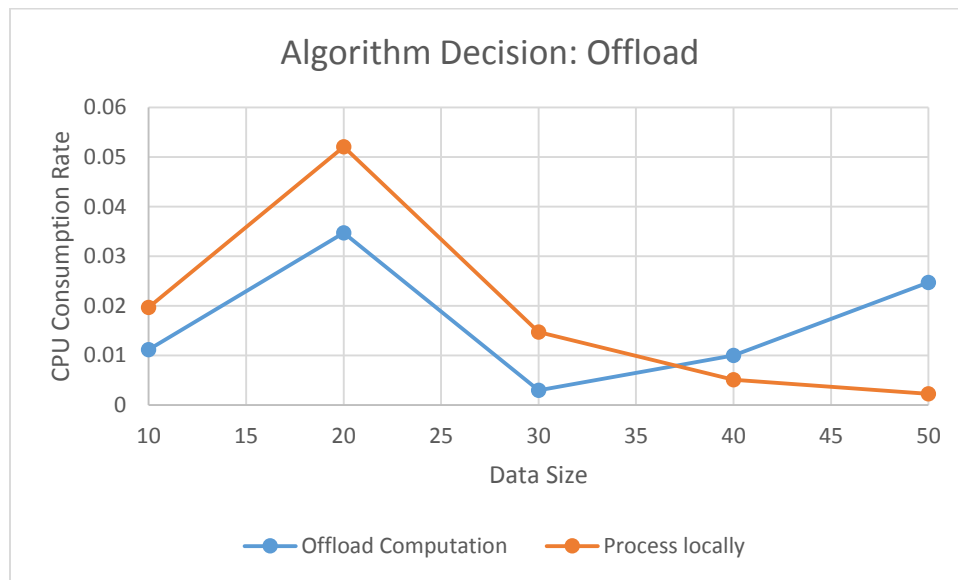


Figure 18: CPU consumption rate for an Offload Decision

5.3 Discussion

From the above experiments, we deduce that our framework is moderately effective in conserving the energy of a smartphone. In Test Case 1, we computed the CPU consumption rate of processing locally when the algorithm decided to offload computation. We subsequently compared that against the consumption rate of offloading at the same time. We can see in the figure above that our algorithm predicts accurately when dealing with small data as offloading truly consumes lower

CPU. However, when dealing with fairly larger data, the algorithm still proposes an offload even though as shown above, that consumes more CPU. This shows that the scheduling algorithm's accuracy in execution time prediction could be further improved. We discuss this in the next chapter.

6 Conclusion and Future Work

Conclusively, in this research study, we examine an approach towards conserving energy on resource-poor smartphones during recurrent, resource-intensive computation. We achieved this first, by examining the prevailing energy-related issues with respect to processing tasks on smartphones and what we intend to resolve through our approach. Secondly, we reviewed current literature and research endeavors with regards to scheduling based computation offloading in order to understand the state of the art of computation offloading and where our approach fits. We also gave an overview of the major technologies we employed in developing our framework. Our contributions in this research study are enumerated below:

- **Enhance Offload Decision Making using a scheduling algorithm:** Our algorithm enhanced the decision making process in offloading scenarios by implementing a fairly accurate execution time prediction methods and comparing them with the communication cost of offloading to ascertain the most suitable course of action with regards to energy conservation during recurrent processing.
- **Propose a Service-Oriented Framework for Energy conservation:** A service-oriented framework was developed towards energy conservation during recurrent, complex processing. The framework was used to implement our algorithm and we developed a proof-of-concept prototype for an Android-based Nexus 5 Smartphone in order to demonstrate the energy-saving abilities of the framework and to prove that the framework can be applied to recurrent-processing of temperature data of varying sizes. The prototype consists of the following entities:

1. Stately.js: A JavaScript finite state machine which we implemented as the backbone of the framework. It initiates and directs the major processes of the framework. It facilitates communication between the external physical computing devices and the framework.
2. Resource Manager: This was the main powerhouse of the framework. Depending on the direction from Stately.js, it executed several functions such as employing the scheduling algorithm in offload decision making and offloading to the local processing service or to the cloud.
3. The framework consisted of a background service that managed the connection and retrieval of context data from the external physical device. It forwarded the extracted data to Stately.js.
4. The task scheduling algorithm in our framework was described as the component responsible for determining the most energy-optimal execution of a task. It was incorporated within the resource manager and it returned a value that indicated the suitability of offloading or local computation.
5. To process context data locally, we implemented a lightweight web servlet container known as I-Jetty that communicated with the resource manager, receiving raw context data and returning it in an understandable format.

With regards to future research directions within the context of this research study, we aim to increase the scope of the framework. Currently, due to resource and time constraints, our scope was limited to the singular scenario of processing temperature data from an external sensor or similar computing device. We would like to augment the framework to be applicable in other ways such as real-time augmented reality visualizations of data from several external physical computing devices. We also intend to add modules to the framework that would allow it to discover, connect and disconnect from specific BLE devices on-the-fly within a given area. We would like to improve the accuracy of our algorithm's execution time prediction function by

employing the use of better, much more advanced machine learning techniques. A minor research impetus would be to optimize the prediction model to predict the execution time of any specified task and not just the temperature parsing scenario we focused on in this text.

Bibliography

- Amazon. (2011). Amazon Elastic Compute Cloud (Amazon EC2). Retrieved from <http://aws.amazon.com/ec2/>
- Balan, R., Flinn, J., Satyanarayanan, M., Sinnamohideen, S., & Yang, H.-I. (2002). The case for cyber foraging. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop: beyond the PC - EW10* (p. 87). <http://doi.org/10.1145/1133373.1133390>
- Brandimarte, P. (2011). Simple Linear Regression. In *Quantitative Methods* (pp. 477–526). <http://doi.org/10.1002/9781118023525.ch10>
- Chen, E., Ogata, S., & Horikawa, K. (2012). Offloading Android applications to the cloud without customizing Android. In *2012 IEEE International Conference on Pervasive Computing and Communications Workshops, PERCOM Workshops 2012* (pp. 788–793). <http://doi.org/10.1109/PerComW.2012.6197619>
- Chen, G., Kang, B. T., Kandemir, M., Vijaykrishnan, N., Irwin, M. J., & Chandramouli, R. (2004). Studying energy trade offs in offloading computation/compilation in Java-enabled mobile devices. *IEEE Transactions on Parallel and Distributed Systems*, 15(9), 795–809. <http://doi.org/10.1109/TPDS.2004.47>
- Chun, B.-G., & Maniatis, P. (2009). Augmented smartphone applications through clone cloud execution. *HotOS'09 Proceedings of the 12th Conference on Hot Topics in Operating Systems*. <http://doi.org/10.1046/j.1365-3083.1996.d01-77.x>
- Cisco. (2014). Cisco Visual Networking Index : Forecast and Methodology , 2013 – 2018. *Middle East, June, 2013–2018*. Retrieved from http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360_ns827_Networking_Solutions_White_Paper.html
- Cuervo, E., & Balasubramanian, A. (2010). MAUI: making smartphones last longer with code offload. *Proceedings of the 8th ...*, 17, 49–62. <http://doi.org/10.1145/1814433.1814441>
- DeCuir, J. (2014). Introducing Bluetooth Smart: Part 1: A look at both classic and new technologies. *IEEE Consumer Electronics Magazine*, 3(1), 12–18. <http://doi.org/10.1109/MCE.2013.2284932>
- Dirksen, J. (2012). Jetty. *DZone Refcardz*. Retrieved from <http://refcardz.dzone.com/assets/request/express/112781/0184982041712a4c99b12\papers2://publication/uuid/5458CD52-650E-4012-B076-C9A3D272D794>

- Fekete, K., Csorba, K., Forstner, B., Feher, M., & Vajk, T. (2012). Energy-efficient computation offloading model for mobile phone environment. In *2012 1st IEEE International Conference on Cloud Networking, CLOUDNET 2012 - Proceedings* (pp. 95–99). <http://doi.org/10.1109/CloudNet.2012.6483662>
- Fernando, N., Loke, S. W., & Rahayu, W. (2013). Mobile cloud computing: A survey. *Future Generation Computer Systems*. <http://doi.org/10.1016/j.future.2012.05.023>
- Fletcher, J. (2009). Multiple Linear Regression. *Measurement*, 338(jan28 3), b167–b167. <http://doi.org/10.1136/bmj.b167>
- Flores, H., & Srirama, S. N. (2013). Mobile Cloud Middleware. *Journal of Systems and Software*. <http://doi.org/10.1016/j.jss.2013.09.012>
- Geoffray, N., Thomas, G., & Folliot, B. (2006). Transparent and dynamic code offloading for java applications. *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, 4276, 1790–1806. http://doi.org/10.1007/11914952_51
- Ghahramani, Z. (2004). Unsupervised Learning. In *Machine Learning* (Vol. 12, pp. 6727–80). <http://doi.org/10.1007/BF00993379>
- Kumar, K., Liu, J., Lu, Y. H., & Bhargava, B. (2013). A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1), 129–140. <http://doi.org/10.1007/s11036-012-0368-0>
- Li, Z., Wang, C., & Xu, R. (2001). Computation offloading to save energy on handheld devices: a partition scheme. *Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 238–246. <http://doi.org/10.1145/502217.502257>
- Love, B. C. (2002). Comparing supervised and unsupervised category learning. *Psychonomic Bulletin & Review*, 9(4), 829–835. <http://doi.org/10.3758/BF03196342>
- Mirashe, S. P., & Kalyankar, N. V. (2010). Cloud Computing. *Communications of the ACM*, 51(7), 9. <http://doi.org/10.1145/358438.349303>
- Mitchell, T. M. (2006). The Discipline of Machine Learning. *Machine Learning*, 17(July), 1–7. <http://doi.org/10.1080/026404199365326>
- Nimodia, C., & Deshmukh, H. (2012). Android Operating System. *Software Engineering*, ISSN, 3(1), 10–13. [http://doi.org/http://en.wikipedia.org/wiki/Android_\(operating_system\)](http://doi.org/http://en.wikipedia.org/wiki/Android_(operating_system))
- Niu, R., Song, W., & Liu, Y. (2013). An energy-efficient multisite offloading algorithm for mobile devices. *International Journal of Distributed Sensor Networks*, 2013. <http://doi.org/10.1155/2013/518518>

- Ou, S., Yang, K., & Liotta, A. (2006). An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems. In *Proceedings - Fourth Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2006* (Vol. 2006, pp. 116–125). <http://doi.org/10.1109/PERCOM.2006.7>
- RStudio. (2012). RStudio: Integrated development environment for R. *The Journal of Wildlife Management*. <http://doi.org/10.1002/jwmg.232>
- Schroth, C., & Janner, T. (2007). Web 2.0 and SOA: Converging concepts enabling the internet of services. *IT Professional*. <http://doi.org/10.1109/MITP.2007.60>
- Su, X., Yan, X., & Tsai, C.-L. (2012). Linear regression. *Wiley Interdisciplinary Reviews: Computational Statistics*, 4(3), 275–294. <http://doi.org/10.1002/wics.1198>
- Sutton, R. S., & Barto, A. G. (2012). *Reinforcement learning*. *Learning* (Vol. 3). <http://doi.org/10.1109/MED.2013.6608833>
- Wen, Y., Zhang, W., & Luo, H. (2012). Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones. In *Proceedings - IEEE INFOCOM* (pp. 2716–2720). <http://doi.org/10.1109/INFCOM.2012.6195685>
- Wikipedia. (2012). Arduino. In *Wikipedia*. Retrieved from <http://en.wikipedia.org/w/index.php?title=Arduino&oldid=517017640>
- Xian, C., Lu, Y. H., & Li, Z. (2007). Adaptive computation offloading for energy conservation on battery-powered systems. In *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS* (Vol. 1). <http://doi.org/10.1109/ICPADS.2007.4447724>
- Yao, D., Yu, C., Jin, H., & Zhou, J. (2013). Energy efficient task scheduling in mobile cloud computing. *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. http://doi.org/10.1007/978-3-642-40820-5_29
- Zhang, W., Wen, Y., & Wu, D. O. (2013). Energy-efficient scheduling policy for collaborative execution in mobile cloud computing. In *Proceedings - IEEE INFOCOM* (pp. 190–194). <http://doi.org/10.1109/INFCOM.2013.6566761>
- Corral, L., Georgiev, A., Succi, G., & Sillitti, A. (2014). Can execution time describe accurately the energy consumption of mobile apps? An experiment in Android. *Proceedings of the 3rd International Workshop on Green and Sustainable Software*, 31-37. <http://doi.org/10.1145/2593743.2593748>
- Masing, K.O. (2013). Predicting the running time of a scientific computation. B.Sc. Thesis, University of Tartu.

Arduino Development Environment. (2015). In *Arduino.cc* Retrieved from <http://www.arduino.cc/en/Guide/Environment>

Wiring. (2015). In *Wiring* Retrieved from <http://wiring.org.co>

Processing. (2015). In *Processing* Retrieved from <http://www.processing.org>

Stately.js. (2015). Retrieved from <https://github.com/fschaefer/Stately.js>

Android Studio. (2015) Retrieved from <http://developer.android.com/tools/studio/index.html>

Android. (2015). Services. Retrieved from <http://developer.android.com/guide/components/services.html>

Android Bluetooth Low Energy API (2015). Retrieved from <http://developer.android.com/guide/topics/connectivity/bluetooth-le.html>

Non-exclusive license to reproduce thesis

I, Oluwaseni Toluwaleke Joshua, (date of birth: 7.11.1992),

1. Herewith grant the University of Tartu a free permit (non-exclusive license) to:

1.1 reproduce, for the purpose of preservation, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright, of my thesis

A Framework for Energy-efficient Mobile Cloud Offloading

Supervised by Satish Narayana Srirama and Chii Chang.

2. I am aware of the fact that the author retains these rights.

3. This is to certify that granting the non-exclusive license does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 21.05.2015.