Execution scheduling methods for mobile applications

Ansuman Dash

Execution scheduling methods for mobile applications

Dissertation submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Computer Science

by

Ansuman Dash
[Roll No: MTC-1305]

under the guidance of

Ansuman Banerjee
Associate Professor
Advanced Computing and Microelectronics Unit

Indian Statistical Institute
Kolkata-700108, India

July 2015

To my family and my supervisor

CERTIFICATE

This is to certify that the dissertation titled **"Execution scheduling methods for mobile applications"** submitted by **Ansuman Dash** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Ansuman Banerjee
Associate Professor,
Advanced Computing and Microelectronics Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Acknowledgments

I would like to show my highest gratitude to my advisor, *Ansuman Banerjee*, Advanced Computing Microelectronics Unit, Indian Statistical Institute, Kolkata, for his guidance and continuous support and encouragement. He has literally taught me how to do good research, and motivated me with great insights and innovative ideas.

My deepest thanks to all the teachers of Indian Statistical Institute, for their valuable suggestions and discussions which added an important dimension to my research work.

I would also like to thank *Dr. Pradipta De*, SUNY, Korea, *Dr. Subhas Nandy*, ISI, Kolkata and *Arani Bhattacharya*, SUNY, Korea for their constructive comments and various discussions on my research.

Finally, I am very much thankful to my parents and my brother, Ayusman, for their everlasting supports.

Last but not the least, I would like to thank all my friends for their help and support. I am especially grateful to Shrabanti, Moumita, Arindam and Dnyaneshwar for always being there for me when ever I had a problem or needed some advice.

Ansuman Dash
Indian Statistical Institute
Kolkata - 700108 , India.

Abstract

Making mobile applications energy efficient immensely builds user satisfaction. Apart from the fact that there are not many efficient techniques for evaluating energy consumption for applications on mobile devices, the methods used are static in nature. Static techniques assume that during the running of an application, no other process can run concurrently, and the concerned application has the entire CPU at its disposal. This thesis is built around three main ideas.

Firstly, we propose a novel idea of measuring the energy consumption of an application running on a mobile device considering the fact that not always the entire CPU is available. This is because the application may sometimes run in the foreground when the mobile is idle and therefore, use the maximum CPU available; at other times, there maybe other tasks being run (apart from the routine background tasks) by the user for which this application is forced to run in the background. The major highlight of this work is in considering the concept of variable CPU availability in energy analysis. We have also suggested to model the energy consumption problem of a mobile phone as a finite state automaton, where our aim is to find if a state can be reached where the entire battery of the mobile phone is exhausted.

As our next work, we address the problem of application scheduling on user mobile devices. Given the fact that a vast number of application may run both as in the foreground and the background, the scheduling task is a challenging one. We propose to solve this problem using clustering and iterative refinement. Results on simulated benchmarks show the efficacy of our proposal.

Mobile Cloud Computing (MCC) offloading has emerged as a key way of mitigating the resource constraints of mobile devices like smartphones. In MCC offloading systems, one or more tasks of the mobile application are migrated and executed on the cloud system. For our final work, we focus on utilizing MCC to optimize applications having tasks with different levels of quality of service. We propose an algorithm to optimize the quality of service of tasks while ensuring that their execution does not exceed the given energy budget. Analysis of our algorithm shows that it provides the optimal solution in polynomial time.

Keywords: *Energy Analysis, Reachability, Finite automaton, Static Analysis, Usage Profile, State Refinement, Static scheduling, Clustering, Mobile Cloud, Quality of Service, Task Scheduling, Linear Work flow, Concurrent Work flow, Topological ordering.*

Contents

List of Figures

Chapter 1

Introduction

In the age where there are tremendous advancements being carried out in the field of mobile phones, the energy limitations of these devices lead to a major area of concern. It is necessary for the users of mobile phones to know the energy constraints of their device so that they can make an informed choice of whether to perform certain tasks or not at a particular time. For this reason, it is necessary to have good energy estimating techniques at the disposal of the mobile phone's user. Recently a lot of research is being carried out in the area of application level energy analysis for mobile applications. One such work has been done in [14], where the authors have designed techniques to monitor application level energy usage on mobile devices. Other works include energy profiling of applications done by taking physical power measurements at the component level on a piece of real hardware [14], or by per-instruction modeling of the application [32].

An important thing to consider while performing energy analysis of an application for execution on a mobile device is the fact that various other applications are in execution simultaneously on the mobile device. All the other applications also consume certain amount of energy while running on the mobile device. Along with this, the user might switch from one application to another at any time. This makes the energy analysis problem of an application that runs on a mobile device a tough one.

Mobile Cloud Computing (MCC) framework enables partitioning power-hungry mobile applications to utilize remote cloud resources. The key technique in MCC is code offloading, which identifies portions of code that are profiled to be computation and energy intensive to execute on cloud servers. The benefit of offloading to save energy on mobile devices has been demonstrated in several prototype systems [18, 16, 38]. However, MCC systems are yet to be in mainstream use on mobile devices [8, 29]. One of the challenges towards practical use of MCC systems is the unpredictable operating environment. There are several sources of variation due to application characteristics, network conditions and platform differences. The unstable bandwidth of wireless networks can hurt the gains derived from

using MCC. Similarly, diversity in application workload on the device, or on the cloud servers can diminish the potential gains from using MCC frameworks.

At the core of it, the problem lies in how the task offloading mechanism in MCC system selects the tasks for remote execution. If the offloading decision cannot factor in the variations in the operating environment, it can lead to poor performance. Several recent works have explored offloading techniques that can adapt at run-time to changes in the operating parameters. Adaptive offloading solutions are complex given the presence of several parameters and their unpredictable variations. Hence the solutions have explored techniques that range from adapting to single parameter to multiple parameters.

A number of surveys have studied MCC systems from various viewpoints. For example, Shiraz et al. discuss different methods of implementing offloading from smartphones [59]. Kumar and Lu show how MCC systems reduce energy consumption on mobile devices [39]. Sharifi et al. present a taxonomy of cyber-foraging systems based on their design and objectives [57]. Another survey, by Sanaei et al., analyzes heterogeneity in MCC systems from an architectural point of view [54].

This thesis is built around three main ideas. Firstly, we propose a novel idea of measuring the energy consumption of an application running on a mobile device considering the fact that not always the entire CPU is available. This is because the application may sometimes run in the foreground when the mobile is idle and therefore, use the maximum CPU available; at other times, there maybe other tasks being run (apart from the routine background tasks) by the user for which this application is forced to run in the background. The major highlight of this work is in considering the concept of variable CPU availability in energy analysis. We have also suggested to model the energy consumption problem of a mobile phone as a finite state automaton, where our aim is to find if a state can be reached where the entire battery of the mobile phone is exhausted.

As our next work, we address the problem of application scheduling on user mobile devices. Given the fact that a vast number of application may run both as in the foreground and the background, the scheduling task is a challenging one. We propose to solve this problem using clustering and iterative refinement. Results on simulated benchmarks show the efficacy of our proposal.

Mobile Cloud Computing (MCC) offloading has emerged as a key way of mitigating the resource constraints of mobile devices like smartphones. In MCC offloading systems, one or more tasks of the mobile application are migrated and executed on the cloud system. For our final work, we focus on utilizing MCC to optimize applications having tasks with different levels of quality of service. We propose an algorithm to optimize the quality of service of tasks while ensuring that their execution does not exceed the given energy budget. Analysis of our algorithm shows that it provides the optimal solution in polynomial time.

Organization of the dissertation

The rest of the dissertation is organized into 5 chapters. A summary of the contents of the chapters is as follows:

Chapter 2: A detailed study of relevant research is presented here.

Chapter 3: This chapter describes a methodology to check if it is possible to schedule a given application on the mobile device. It also introduces the concept of variable CPU availability for applications running on the mobile device.

Chapter 4: This chapter proposes refining the finite state automaton created in the previous chapter.

Chapter 5: This chapter introduces the concept of variants of the tasks that run on the mobile device and on the cloud architecture.

Chapter 6: This chapter presents an overview of the thesis and the future work to be carried out.

Chapter 2

Background and related work

In this chapter, we first present a few background concepts needed for developing the foundation of our framework.

2.1 Energy Analysis

Every mobile device has its own energy limitations. The energy constraints of the device should be known to its user; for this, good energy estimating techniques are required. In recent times, there has been a lot of research for designing techniques to monitor application level energy usage on mobile devices. Energy profiling of applications can be done by taking physical power measurements at the component level on a piece of real hardware [14], or by per-instruction modeling of the application [32].

Most of the energy analysis methods existing in literature have dealt with applications which are expected to run in the foreground, and therefore, the energy estimate is usually a conservative over-approximation of the actual energy footprint at run-time. This is because of the fact that these methods typically attempt to infer the longest energy path in the application for any given input, while at run time, such inputs may not occur in practice and the actual energy consumed maybe less than what is estimated.

In recent times, several articles have provided various models to predict the power consumption of a mobile application. One of the ways to carry out energy analysis is through program analysis. In a paper by Hao et. al [32], a novel approach to the energy analysis problem is proposed. They have tried to calculate the code-level estimates that an application will consume at run-time by analysing the implementation of it on the mobile device. Furthermore, they also summarized the results at the granularity of the whole program, path, method, and source line which can enable reduction of energy consumption of the application.

A novel methodology to monitor average permanence with CPU availability has been proposed by Banerjee et. al in [7]. Thompson et. al in [61] have suggested a model-driven methodology to evaluate the power consumption of any mobile application architecture. They have proposed the System Power Optimization Tool (SPOT) which predicts the power consumption of an application architecture by generating device logic. This device logic can be used during early phases of an application's software life-cycle to gather power consumption information on physical hardware. Carroll et. al in [14] have analyzed the power consumed by the hardware components of a mobile phone. For doing this, they have done the physical power measurements at the component level on a piece of real hardware. Bhargava et. al in [10] have tried to come up with the power consumption characteristics of various distributed and centralized data mining algorithms that can run on a mobile device.

Pathak et. al in [49] have suggested an implementation of a fine-grain energy profiler for smartphone applications. Ding et. al in [23] have given a smart energy monitoring system that can profile mobile applications with battery usage information. Flinn et. al in [28] have calculated the energy consumption of mobile applications by mapping it with program structure. Murmuria et. al in [46] have used the per-subsystem time shares reported by the operating system's power-management module to effectively account for the power usage of all the primary hardware subsystems on the phone as well as individual applications.

2.2 Scheduling of tasks

Scheduling is instrumental in achieving high performance in parallel and distributed systems. Work on static multiprocessor scheduling dates back to 1977 [60], where the problem of scheduling a directed acyclic graph of tasks on two processors is solved using network flow algorithms. Further research in this direction focused on scheduling distributed applications on a network of homogeneous processors [42]. As optimal multiprocessor scheduling of directed task graphs is an NP-complete problem [47], heuristics are vastly used. A wide range of such static scheduling heuristics have been classified and rigorously studied [12, 40, 45].

The idea of partitioning the task graph is fundamental to our job abstraction. Many existing heuristics already use the idea of clustering groups of tasks in the task graph to simplify the scheduling problem [22, 31, 65]. However, we are not aware of any existing scheduling heuristics that applies the same principle to the data center representation, or that systematically explores the idea of pessimistic abstraction to get coarser partitions and, thus, improved scalability. Also we are not aware of any clustering heuristic that uses a refinement loop to change the partition and increase the quality of the produced schedules.

Many generic heuristics for solving optimization problems such as genetic algorithms and simulated annealing have been used for scheduling [35]. Like our technique these techniques iteratively search for local optimal solutions but directly solve the concrete problem instance and do not use abstraction. While these generic approaches produce good schedules, their

performance is rather poor [12].

Systems like Hadoop (Apache Hadoop) and DryadLINQ [66] use dynamic scheduling techniques in favor of static scheduling because they are designed for environments with incomplete information about both the requirements of executed jobs and the available resources. Hybrid approaches that combine static and dynamic scheduling can help to increase performance even in incomplete information environments [41]. Our framework provides many opportunities for exploring such hybrid approaches. By design, our schedulers already work with incomplete information. One can use the idea of a scheduling horizon [21] where only tasks that are to be executed in the immediate future are dispatched to their scheduled nodes. For tasks that are to be executed later one can then compute abstract schedules that only provide an approximate plan for their execution. These abstract schedules can be refined dynamically as more precise information about depending task becomes available. Schedulers like FISCH and the BLIND scheduler work hierarchically, so they can be decomposed into different levels, which enables distributed scheduling. Finally, both schedulers can easily integrate dynamic scheduling techniques such as backfilling [43].

2.3 Clustering

Clustering can be considered the most important unsupervised learning problem [52, 67]; so, as every other problem of this kind, it deals with finding a structure in a collection of unlabeled data. A loose definition of clustering could be “the process of organizing objects into groups whose members are similar in some way”. A cluster is therefore a collection of objects which are “similar” between them and are “dissimilar” to the objects belonging to other clusters. The term cluster analysis (first used by Tryon, 1939) encompasses a number of different algorithms and methods for grouping objects of similar kind into respective categories. A general question facing researchers in many areas of inquiry is how to organize observed data into meaningful structures, that is, to develop taxonomies. In other words cluster analysis is an exploratory data analysis tool which aims at sorting different objects into groups in a way that the degree of association between two objects is maximal if they belong to the same group and minimal otherwise. Given the above, cluster analysis can be used to discover structures in data without providing an explanation/interpretation. In other words, cluster analysis simply discovers structures in data without explaining why they exist.

2.4 Mobile Cloud Computing

A user expects the mobile system to run a variety of applications. However, a mobile system is constrained by the residual battery capacity at any point in time, and the limited computation power of existing mobile processors. Mobile Cloud Computing (MCC) or

cyber-foraging aims to enable energy or computation intensive applications on mobile systems by distributed execution of mobile applications. This is done by process offloading, i.e. migrating a portion of the application state from the mobile device to remote computation resources.

The user executes a variety of mobile applications on the smartphone. The smartphone can access the Internet through one or more wireless network channels. It migrates some portions of the running applications to other computation resources available using the network. Such computation resources may be the user's own personal computer, a processor attached to an wireless access point, other mobile devices, a cloud server or even routers and switches in the network. On completion of the task on the remote server, the new program state is transferred back to the smartphone.

In order to enable distributed execution of a mobile application, the MCC system must determine how to partition an application for scheduling on mobile device and cloud servers. This is decided by the offloading decision engine, which may be present either on the smartphone or on a pre-defined server. The offloading decision engine needs to identify the most energy or computation intensive tasks of the given application. Using this information and the condition of the environment, it selects the part of the program to be offloaded for remote execution. The decision taken by it determines the amount of resources saved on the mobile device and the quality of experience (QoE) of the user.

A mobile system is used in a variety of conditions by the user. For example, users can move while talking to someone on the phone, or view videos while sitting in a car. This affects the network performance. Similarly, the user might switch to another application, and thus change the workload on the MCC system. The computation power of the remote resources and the mobile device can also vary.

Performance of MCC systems are affected by these changes in operating conditions. MCC systems are affected by wireless network characteristics and the capabilities of available computation resources. The workload also varies depending on the applications that are running on the smartphone. An MCC system has to handle these changes while maintaining a good QoE for the user. In order to build an adaptive MCC system, a proper understanding of the range of environmental variables and user expectations is essential. The architect of an MCC system can then incorporate the user expectations while implementing it. To meet the user expectations, the use case scenarios can help in determining the importance of different parameters for adaptation.

2.5 Handling Application Variations

A large variety of mobile applications are run on mobile devices. The expectations of the user from each of these mobile applications vary. For example, a user expects an application having lot of interaction to have fast response time. However, for a computation-

intensive application running in the background, it is more important to reduce its energy consumption. An MCC system needs to handle these multiple applications and sometimes the conflicting expectations from the users. In this section, we discuss how MCC systems handle the variation in applications. Variation within applications affect all aspects of the user's quality of experience (QoE).

2.5.1 Concurrency

The amount of parallelism that can be exploited increases with an increase in the concurrency of an application. Thus, applications having higher amount of concurrency tend to give better performance using MCC [38]. However, the offloading decision problem for a sequential application is much more scalable. Moreover, there can be different types of concurrency in an application – at the task (or method) level, and at the data level. The type of concurrency available for an application also has an influence in the performance of an MCC system.

We first discuss the scalability issues related to concurrency. We then discuss how the type of concurrency influences the design decision of an MCC system.

Improvement in scalability has two major advantages. First, a highly scalable solution allows the offloading decision algorithm to run on the mobile device. Secondly, it allows the MCC system to take a more optimal decision. Both these advantages lead to energy and time savings.

The offloading decision problem is polynomial for sequential applications [44]. For such cases, the decision problem formulation can be resolved to the shortest path problem. This allows an MCC system to schedule larger applications in less time.

For applications having task-level concurrency, the offloading decision problem is known to be NP-Complete [62]. Thus, to build more scalable techniques for concurrent applications, efficient algorithms to partition such applications are required [17]. One heuristic used by ThinkAir is to compare the amount of computation involved for each thread and the migration costs [38]. If the local computation cost exceeds the migration cost, only then execution is performed on the cloud system using migration. Another work, Hermes gives a polynomial approximation algorithm for applications with limited amount of parallelism [36].

Applications having data-level concurrency have more flexibility in execution. For some applications, like computer vision, the output precision reduces with an increase in the number of threads. This reduction in precision is acceptable upto a certain level. Thus, the MCC system may decide the optimal level of concurrency. Odessa is an example of such an MCC system [50].

2.5.2 Multiple Applications

Most MCC systems developed so far have discussed only the execution of a single application at a time. Smartphone devices have multi-tasking operating systems and usually run multiple applications at the same time.

MCC systems need to model multiple applications for dealing with realistic use cases. This is because, the energy consumption involved in communication depends not only on the amount of data, but also on the time when the network interface was last used. This occurs because interfaces of radio networks remain switched on for a few seconds even after transmission is complete. A request for transmitting data takes less energy if the network interface is already on. However, if no such request is sent, then the energy required to keep the interface running is wasted. Experimental results show energy saving upto 60% by careful timing of communication requests when radio networks are used [56].

Studies have shown that finding the optimal solution for multiple applications is NP-hard [20]. One such study has proposed using coalesced offloading, where the algorithm tries to consolidate the network transmissions of different applications. Experiments on real-world use cases have shown energy saving of 21% over naive scheduling [64].

2.6 Handling Execution Platform Variations

The underlying architecture of the MCC system is an important consideration in the design of system architecture. Moreover, the architecture of the mobile devices as well as that of the cloud system is constantly evolving. This makes it even more important for an offloading framework to adapt to different systems available in the market.

2.6.1 Mobile Processors

Modern smartphones have up to 8 processors. An increase in the number of mobile processors reduces the completion time of an application, provided it has sufficient parallelism. However, it increases energy consumption.

The first MCC systems developed assumed a single processor system [18, 16]. Recent works have shown that modifying the algorithm to optimize the completion time for multiprocessor systems is possible [5]. However, the effect on energy consumption is still not clear due to lack of a relevant energy model.

In order to limit the amount of energy consumption, mobile processors now have an inbuilt scheme of running at different frequency levels. This is known as Dynamic Voltage and Frequency Scaling (DVFS). In this scheme, a processor that runs at lower frequency takes

more time to execute a task, but consumes less energy. It allows the application scheduler to suitably decrease the energy consumption at the cost of higher completion time.

An MCC system must determine the number of frequency levels that each mobile processor supports. It needs to have an algorithm that can adapt to the possible frequency levels. Although some studies have proposed algorithms to adapt to multiple frequency levels, they have not been implemented in real systems.

Balakrishnan and Tham propose an optimization formulation to partition the application considering the possible frequency levels [5]. Chen et al. show that the battery discharge rate depends on the interaction between the strength of wireless signal and the DVFS level of processors [15]. Thus, the energy consumed by these components must be considered together in an MCC system. These works have been evaluated using simulation. There are some works in the literature showing that simulation of DVFS overestimates the amount of energy saving [53]. This is because modern processors spend a large component of the power in idle state. Thus, an implementation of an MCC system is necessary to evaluate the actual energy saving using DVFS.

2.6.2 Time and Energy Profile of Applications

The energy consumed by the execution of an application is affected by the type of hardware used. This includes the type of processor used, the presence of the coprocessor, the type of network interface and the capacity of the battery. For an MCC system to take correct decisions, it must be able to adapt to these changes in hardware without any manual involvement.

Most MCC systems, such as [19, 16, 38], depend on energy models to determine the energy consumption of an application. These energy models compute the energy consumption by looking at either the source or the intermediate code of the application. Thus, these systems developed their own energy models using specialized high-precision power monitors. However, this is only feasible on a limited number of different mobile devices. For wide use of offloading, a more automatic system of estimating energy is needed.

Studies have shown that an accurate energy model can be developed automatically. The tool PowerBooter develops an automatic energy model for each phone [68]. It calibrates the energy model by running a specific hardware component on the mobile device over a period of time. The difference in voltage can be used to compute the amount of energy consumed by looking at the discharge rate curve available in the battery's manual. While this can handle the variation in mobile hardware, the power model for each battery has to be calibrated manually.

Sesame shows a method of handling multiple batteries automatically [24]. It uses the registers exposed by the battery's interface to measure the remaining energy. This can be

used to study the applications running, and to find their energy profile over a period of time. In this way, an energy model can be built automatically for each mobile device and battery.

One problem with these profiling techniques is that they do not consider the impact of user input on the execution pattern. The user input plays an important role in determining which methods of an application get executed. Thus, it has a major impact on energy consumption. Gao et al. model these variations using a semi-Markov model to arrive at a more accurate estimate of the energy consumption [30].

2.6.3 Cloud architecture

The rapid increase in the number of available computing devices makes offloading to them a feasible option. This is especially true for latency-sensitive applications, where increase in latency has a major impact on quality of experience (QoE). Thus, reducing latency by offloading to devices lying nearby, thereby saving local compute energy, might be more attractive.

One proposed way of lowering latency is to maintain computation resource closer to the mobile device, known as cloudlet. Unlike a cloud server, a cloudlet is located at a one-hop distance from the mobile device. However, it has less computation power compared to a cloud server. Cloudlets can be attached to wireless access points to enable easy access from mobile devices.

In order to handle applications with real-time constraints, multiple hierarchy of cloud systems has been suggested. This includes using a combination of cloudlet and cloud systems. Satyanarayanan et al. show that cloudlets can help applications improve performance in hostile environments, such as war and disaster-relief [55]. Zhang et al. show that the QoE can be improved using both cloudlets and cloud systems as compared to just one of them [69]. MAP-Cloud proves that finding the optimal solution to determine the execution points of components is NPhard [51]. It proposes some heuristics that can be used to partition the application to save energy. CARMS uses locationing to select the best combination of cloud resources [37]. However, these approaches require installation of additional infrastructure, since cloudlets are not readily available on wireless access points [3]. A multi-level hierarchical mobile cloud system requires a relevant pricing model. So far, most studies have not considered the cost that the user has to pay to the cloud service provider. For widespread adoption, a pricing scheme acceptable to both the user and the cloud service provider is needed.

Another technique of reducing latency is utilized by CDroid [8]. CDroid tightly integrates the cloud server to a mobile device. It sends the state information of an application when the mobile device is idle. When a computation-intensive request is received by the mobile device, most of the information required to handle is thus already available on the server.

In this way, it utilizes a dedicated server to cache application state information in order to reduce the amount of data transmission.

One offloading technique proposed is utilization of other mobile devices available nearby. The advantage of this approach is that at a particular point, a large number of mobile devices are usually available. Misco [25] offloads data-intensive applications in order to enable data-level parallelism. Serendipity implements fine-level offloading at task level to nearby mobile devices to speed up execution [58]. ECC investigates the devices to which offloading is attractive based on their proximity, current situation of battery and processor, and user objective [9].

Offloading to other mobile devices provides a relevant economic incentive to the users of the remote devices. Execution on the remote devices consumes energy, reducing their battery life. One study has shown using mathematical modeling that cooperation among remote mobile devices improves the battery life of all of them [63]. However, this assumes that the same mobile devices are always available in the system.

Although utilization of other mobile devices has potential, it carries a high security risk. This is because, it is possible for a malicious remote device to permanently store the user's data. It might also be possible for the remote device to make the mobile application more vulnerable to security threats. Abolfazli et al. suggest sandboxing and signing of the remote system to detect any modifications [4]. However, the security of such a system has not been tested yet.

A recent proposal suggests offloading to network devices such as routers and switches. This is known as fog computing [11]. This does not require new infrastructure, and also has low latency. However, the challenge is to handle the very high level of heterogeneity among the different network devices. Mobile fog proposes a programming model to enable fog computing for mobile devices [34]. The primary challenge of using fog computing is in the handling of architectural heterogeneity. Conventional frameworks that support architectural heterogeneity like openCL are not supported by network devices [26]. So far, no MCC system is available that utilizes fog computing.

2.7 Impact on Quality of Experience (QOE)

The variations in the environmental parameters affect the quality of experience (QoE) of the user. Although the objective of the MCC system is to maintain high QoE in a difficult environment, this is not always possible. In this section, we study the parameters that affect the QoE of the user. We also briefly discuss how the importance of different QoE parameters may vary depending on the context.

2.7.1 Energy Consumption

We first discuss the effect of the application component on energy consumption. Increasing the amount of concurrency and multitasking allow higher energy savings by making it easier to offload software components. However, increase in real-time constraints increases amount of local execution, and reduces energy savings.

The network component of the ecosystem is also an important factor influencing energy consumption. Higher bandwidth reduces the energy cost of transmission, thus saving energy. Intermittent connectivity leads to checkpointing of applications, and more local execution, thus leading to lower energy savings. Latency has no effect on energy, since it only affects completion time. Network interfaces have different energy consumption patterns, and thus the energy saving varies. Greater user mobility increases energy consumption and lowers energy gain by increasing the cost of hand-off from one base station to another [6].

The execution platform also affects the energy consumption. Increase in number of mobile processors increases the energy cost, and lowers energy savings. Intelligent use of Dynamic Voltage and Frequency Scaling (DVFS) can lead to higher energy savings, by allowing a processor to run at lower speed, while offloading more tasks. Having a coprocessor also leads to lower energy savings, since running it increases the energy consumption. Having a higher energy profile gives the MCC system more scope to offload, and leads to higher energy savings. A cloud architecture having more computation resources nearby reduces the energy cost of transmission, and thus leads to higher energy savings.

2.7.2 Completion Time

The completion time is affected by the number and type of applications given by the user. Higher concurrency and more multitasking allows better utilization of parallelism, leading to greater time saving. Real-time constraints force the MCC system to schedule the tasks in a way that satisfies the constraints. This may lead to higher or lower completion times, depending on the condition of the channel in the MCC system.

The network condition has a major impact on completion time. Higher bandwidth and lower latency lead to faster transmission of data, and thus greater time saving. Intermittent connectivity leads to loss of contact with the cloud system, leading to more local execution and thus lower time saving. Some network interfaces inherently provide higher bandwidth and lower latency, thus affecting the completion time. Higher user mobility forces the cloud system to migrate data across different servers, in order to better support the user. This leads to lower time saving.

The execution platform also affects the completion time. Increase in the number of mobile processors leads to faster execution, but decreases the scope of offloading, and thus reduces time saving. A similar result is obtained by adding a coprocessor on the mobile system.

However, DVFS has no effect on the completion time, unless the MCC system optimizes energy consumption.

More computation-intensive jobs give the MCC system more scope to offload, leading to higher time saving. A cloud system with more layers and having some layers closer to the mobile device reduces latency and thus leads to higher time saving.

2.7.3 Monetary Cost

User surveys suggest that controlling the monetary cost incurred by mobile software is the greatest concern of users [27]. Thus, an adaptive MCC system has to control the monetary cost. The application component of the mobile ecosystem has a significant impact on the monetary cost. Utilizing higher amount of concurrency and more multitasking require more server processors, thus increasing the server cost. Having more real-time constraints increases the amount of communication, leading to higher bandwidth cost.

The monetary cost of offloading is affected by only the network interface used by the MCC system. This is because network interfaces have different pricing models. For example, since 3G has lower bandwidth and higher latency than LTE, data transfer over 3G usually has a lower cost. Thus, using an interface that provides better service increases the monetary cost.

Among the many parameters of execution platform, only the cloud architecture affects the monetary cost. This is because hiring more computation resources is expensive. Thus, the cost goes up with an increase in more layers of cloud system.

2.7.4 Security

Mobile devices usually contain a lot of private data of users. Moreover, ensuring the integrity of the data that has been computed remotely is also important. Thus, security forms an important component of the QoE.

The type of application has an impact on privacy of data. Having more real-time constraints in the application increases the number of migrations from the cloud to the mobile device. This makes the system more vulnerable to a man-in-the-middle attack, where the attacker alters the packets during transmission. The variations in the parameters of the network component also affect security. Some network interfaces, like Wi-Fi, are more vulnerable to man-in-the-middle attacks, as compared to 3G or LTE [48]. Higher mobility of user increases security by making it more difficult to sniff data [13].

Only the cloud architecture in execution platform component has an impact on security. A malicious remote computation resource used by the mobile device for offloading can

subtly modify the execution of the application. This can affect the integrity of the running application, making the system vulnerable to attacks.

2.7.5 Novelties of this thesis

In this thesis, we have suggested a more realistic energy analysis methodology for applications and have considered variants of the applications that are to be executed. Along with energy analysis which is done statically, varying percentage of CPU of the applications running on the mobile device has been considered. Applications are monitored as state transition automaton and is refined to meet our objective which is discussed later in this thesis. We also analyze the use of multiple variants. Multiple variants of a single task leads to multiple possible energy paths, and the scheduler can choose an energy path based on the energy-constraint.

Chapter 3

REAST: Residual Energy Aware Scheduling Technique for mobile applications

In this chapter, we deal with applications (e.g. Play Store update, virus scan, etc.) which may run as both, a foreground task when the mobile device is idle and as a background task when a user application is triggered in foreground. In such cases, a straightforward application of the energy analysis methods existing in literature may lead to under-estimates of the total energy consumption. This is because of the fact that these methods [32] typically assume that the application will have the maximum CPU at its disposal (except the routine system processes) when running in the foreground of the mobile device. However, in reality, for the kind of applications we consider, there is a considerable amount of interaction with the user, and in all the ways the mobile is used, full CPU power may not be available for running the application which may have to run as a background application with varying percentages of CPU available. For example, when the mobile device is idle, the concerned application may have the maximum portion of the available CPU, however, this reduces when the user attends to a call or listens to music and the application is forced to run in the background causing the application to take some extra time than estimated. This additional time is caused by the foreground application started by the user, in addition to the routine jobs running on the mobile device. While the energy consumption due to the routine background jobs maybe available from kernel measurements, the battery consumption due to the foreground jobs is difficult to estimate apriori.

Scheduling such classes of applications therefore have to be done carefully, given the current energy level of the mobile device and the application running in the foreground. Standalone application level energy analysis of a Google Play Store update, for example, may give us some estimate of the battery to be exhausted as a result of an update invocation. However this is only a lower bound of the actual energy consumed at run-time, considering the

fact this update will run in context switching mode, both as a foreground and background application on the mobile device. Thus, whether an update invocation leads to the battery getting depleted completely is a question to the user. Considering other processes running on the device when we start an application, we need to decide if the phone will ever reach a situation such that the entire battery is exhausted. This problem is the major point of discussion in this chapter. In particular, there are two major highlights of this proposal as outlined below.

- We propose the notion of an application usage profile model for modelling the energy footprint on a mobile device. This model is built from usage logs of a given mobile device, monitoring the energy consumptions per unit time for each running application and tracking different kinds of applications run by the user while also monitoring the times for which they are run.
- Given the usage profile as above, we attempt to solve the energy analysis problem in a more realistic setting compared to the methods existing in literature.

3.1 Motivation and Objectives

We take our motivation from the work done by Hao et. al. [32]. In this paper, the authors have proposed a novel approach for calculating the energy consumption of a mobile application through program analysis. They have suggested a tool and have extracted as its output, the code-level estimation of the energy that an application will consume at run-time, through analysis of the implementation of the application.

We now argue why the method above is an under-estimate and maybe misleading in practice. Consider that we wish to perform a Google Play store update on a smartphone. The energy consumption of the update can be predicted by the model proposed in [32] which does not talk about the percentage of CPU that is available for Play store update and assumes that at all the times, the CPU available is maximum available to a foreground process. We now analyse the situation more realistically. Let us assume that the update needs to install a 2MB patch for which the estimated energy consumed will be 20J with 1J of energy being consumed in each second. This energy consumption is applicable if we have 100% CPU for the use of the Google Play store update. This always might not be the case and if the update runs as a background job, the actual battery consumed will be much more than 20J since during the entire duration, the user might run various other applications in the foreground.

Since the value of energy consumed per second with 100% CPU for the update is 1J and it will consume in total 20J of energy, it will run for 20 seconds. Assume that 20% CPU is available for its use at the time when the update installation begins. Then the update after starting will have its new value of energy consumption per second as 1* 20/100 =

0.2J. So, to do its entire task with just 20% CPU, it will take 20/0.2 = 100 seconds. Thus, the total battery consumed by the mobile phone for the entire duration of 100 seconds will be the total energy consumed by the update and the application which was running in the foreground during this period considering that the device stays in the same application state in this period.

In practice, the above situation may not be true and the mobile device may switch between multiple applications and therefore, the update installation may actually witness various fractions of CPU availability during its operation. Moreover, the amount of time for which the device stays at a certain state is also variable and user dependent. Given such a situation, energy analysis becomes a difficult problem as we describe below. To capture the application level variability, we present here the notion of an application level usage profile model, typically collected from user usage logs. To handle the variability in time for which the user runs an application we use the notion of average permanence as described below.

Consider Figure 3.1 which shows three applications that are obtained from the log dumps of an android mobile phone user, and two blocks representing the phone when in idle state and when it is switched off. The average permanence time (H) and the rate of energy dissipation (R) are also shown corresponding to each state of the user. Rate of energy dissipation includes the energy consumed by the system processes. Average permanence is the amount of time the user uses a particular application in one go after starting the application. In other words, it is the time difference between starting an application and ending it. Monitoring the average permanence with CPU availability has already been done as a probabilistic finite state machine by [7]. We augment the same model and use it in our work as described in Section III. We now describe how to obtain the energy footprint of the Play store update given the above model. Let us assume when the Google Play application starts, the user is in the Youtube state, with a battery of 250J left and the user has been using Youtube for the last 900 seconds. According to the permanence time of Youtube as given in Figure 3.1, the user will stay in that state for another 35 seconds. In this time, let us assume that the update is getting 20% CPU for its use. So, for the next 35 seconds, the total energy consumed will be 4.15 (for Youtube) + 0.2 (for update) for 35 seconds, that is, 145.25J + 7J = 152.25J. Now, the Google Play store update needs another 20J - 7J = 13J to complete its task. The user now makes a transition to the idle state where the update is getting 90% CPU for its use. The amount of work done per second will be 1* 90/100 = 0.9J, for which the update will take 13/0.9 = 14.45 seconds. The energy consumed is 3.34 (for Idle state) + 0.9 (for update) for 14.45 seconds = 48.26J + 13J = 61.26J. So, the total energy consumed is 152.25J + 61.26J = 213.51J, which is less than 250J initially available with the mobile device. Thus, the update can be allowed to begin. If instead of going to idle state, the user would have played Temple Run 2 and CPU available for the update is, say, 40%, the amount of work done by the update will be 1* (40/100) = 0.4J per second. It will take another 13/0.4 = 32.5 seconds to complete its task. The energy consumed is 3.47 (for Temple Run 2) + 0.4 (for update) for 32.5 seconds = 112.78J + 13J = 125.78J. In this scenario, the total energy consumed is 152.25J + 125.78J = 278.03J which is more than the 250J that was initially available for the mobile device. Thus, the update should

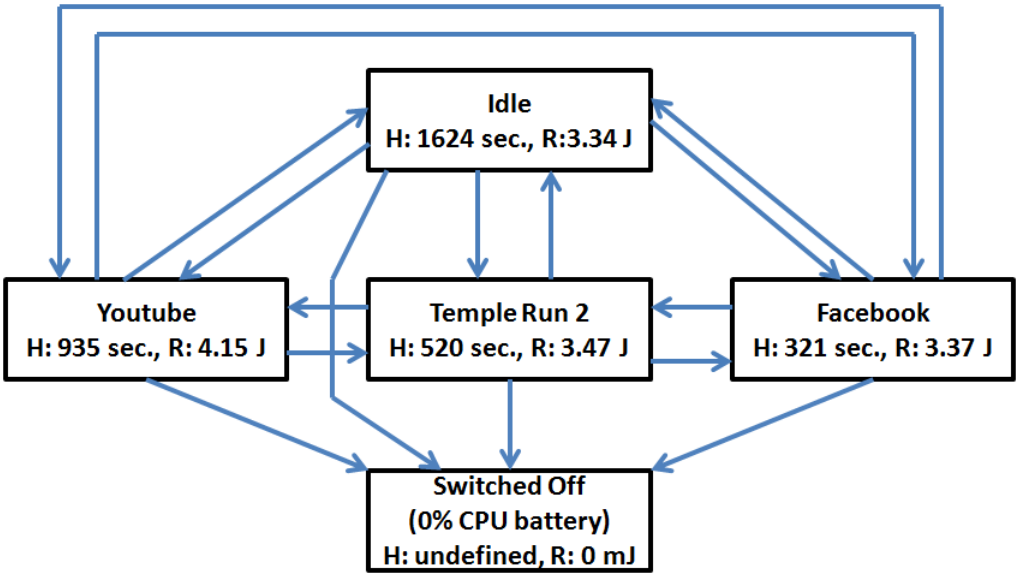

Figure 3.1: Usage Profile Automaton (UPA)

not be allowed to start.

This example scenario motivates our proposal. It is necessary to model the usage profile while performing the energy consumption estimation during the entire computation, else, we may end up with an under-estimate that may not hold in reality. The following section illustrates our proposal of usage profile modelling and energy consumption analysis.

3.2 Formal Model

We now formally define the notion of the application level Usage Profile Automaton (UPA). A number of research articles such as [7] have analyzed and modelled usage profiles on smartphones and we intend to use that for our analysis. The automaton will be represented with the following parameters:

- S, denoting the set of application states $\bigcup$ the (idle, switched off) states.
- E is a set of edges with one edge present between each state pair. In other words the automaton is a graph without self loops
- H: S $\rightarrow \mathbb{R}$ represents the average permanence value defined as H(s_i) = t_i, where $s_i \in$ S, $t_i \in \mathbb{R}$.
- R: S $\rightarrow \mathbb{R}$, denoting the rate of energy consumption at a state.

- C : S $\rightarrow$ [0%, 100%] is the CPU availability.

Each foreground application (i.e., applications which run only as foreground jobs) that the user runs on his device appears as a state in S. The rate of energy dissipation also includes the contribution of the routine system processes. As discussed earlier, the focus of our analysis is more on mixed-mode applications (i.e., applications that run as foreground jobs when the CPU is idle and as background jobs when some other application is running on the mobile device).The problem we address is as follows. Given a state $s \in S$ and remaining battery level l, is it safe to run a mixed mode application A with energy requirement E_A given that the user has already spent some time in the state s, and may transit to any other state after the average permanence time for that state is over. To answer the question above, we need to check if the zero battery state is reached by the time this mixed-mode application finishes. We answer the previous question with the help of a simple algorithm on our Usage Profile Automaton. Let the time remaining in the current state s be t_r which is obtained by subtracting the time elapsed from the average permanence of s. Algorithm 1 answers the reachability problem. Let F be the critical state.

Algorithm 1 Algorithm for reachability

```
1 Input: UPA, l, F (critical state), s (current state), t_r. (time elapsed in the current state), E_A (Total energy
required by the mixed mode application to complete its operation)
2 Output: F reachable or F not reachable
3 FUNCTION FindPath (s, E_A, t_e)
4     WHILE all possible paths from current state s have not been explored
5         Get C_A = C(s)
6         IF (C_A X R_A X t_r) < E_A
7             THEN E_A' = E_A - (C_A X R_A X t_r)
8                     make transition to the next state S'
9         ELSE return "F not reachable"
10              EXIT
11        ENDIF
12        IF the new state S' is F
13            THEN return "F reachable"
14                 EXIT
15        ENDIF
16        FindPath(S', E_A')
17    ENDWHILE
```

Algorithm 1 essentially does a depth first traversal beginning with the current status and for each new state encountered in the iteration at every level, it tries to check if the critical state is reached. If at any level this is any true, the algorithm exists and return that F is not reachable. Otherwise, this leads to a recursive call to a new state. In this manner, every path beginning with the current state is explored by our algorithm. At the end, we can get a more realistic estimate of the energy consumption for the mixed mode application. If the critical state indeed turns out to be reachable, we can advise the user not to start the mixed-mode application.

Extending the energy analysis with transition probabilities

The algorithm bound provides a conservative analysis towards the reachability problem. If the zero battery state is reachable by any path from the current state, the algorithm will advise not to use the application. However in reality, this path may have very low probability of occurrence. To capture this, the idea of Usage Profile Automaton can be extended to a probabilistic Usage Profile Automation where the edges are labelled with probabilities of those transitions occurring. This can be obtained by logging user usage patterns on smart-phones and applying supervised learning to build the state-transfer function. The state diagram of such a automaton is similar to the one shown in Figure 3.1 with transitions annotated by transition probabilities. With this additional information we can annotate each path obtained by Algorithm 1 with corresponding probabilities of occurrence. This can be of greater help to the user. For low probability paths, reaching the critical state may not be as important and the user can take an informed decision.

3.3 Experimental Results

To prove that the applications that we intend to run, in our case the Google Play store update, do not get 100% CPU at their disposal, we used energy measurement applications available in Google Play store to calculate the actual energy consumed by our application when it runs on the mobile phone. There are various applications that do this modelling like [1, 2]. These applications give us the amount of CPU being used by various processes running on the mobile phone along with their energy uses. From their log files, we can get all the applications that had run for a given period and the energy each of them consumed in each second of time. Consider the snapshot taken from such an app as shown in Figure 3.2. It is clear that Google Play store, which has been running for the last 40 seconds, has consumed 5.2J of energy and is currently using 14.8% total CPU processing capacity. There are various other applications running simultaneously on the mobile phone such as Google Contacts Sync and PowerTutor2 which are also utilizing some portion of the CPU. A lot of the CPU is being used by the system processes that are running on the phone. Thus to assume that 100% CPU is available for our concerned application is not quite correct.

We took some data readings with a time period of 10 seconds between each reading. The data consist of the power consumed by different processes running on the mobile phone at those time instances. The total energy consumed is the sum of the energy consumed by the LCD, the CPU as well as the network communication used for all the processes for a duration of 100 seconds. We considered to monitor these three parameters because all three of these consume the battery of the mobile phone. We also calculated the amount of energy consumed by the Google Play store alone in intervals of 10 seconds. Finally, we compared the two set of values in the form of a graph as shown in Figure 3.3. The x-axis represents the time elapsed in seconds and the y-axis represents the energy consumed in Joules.

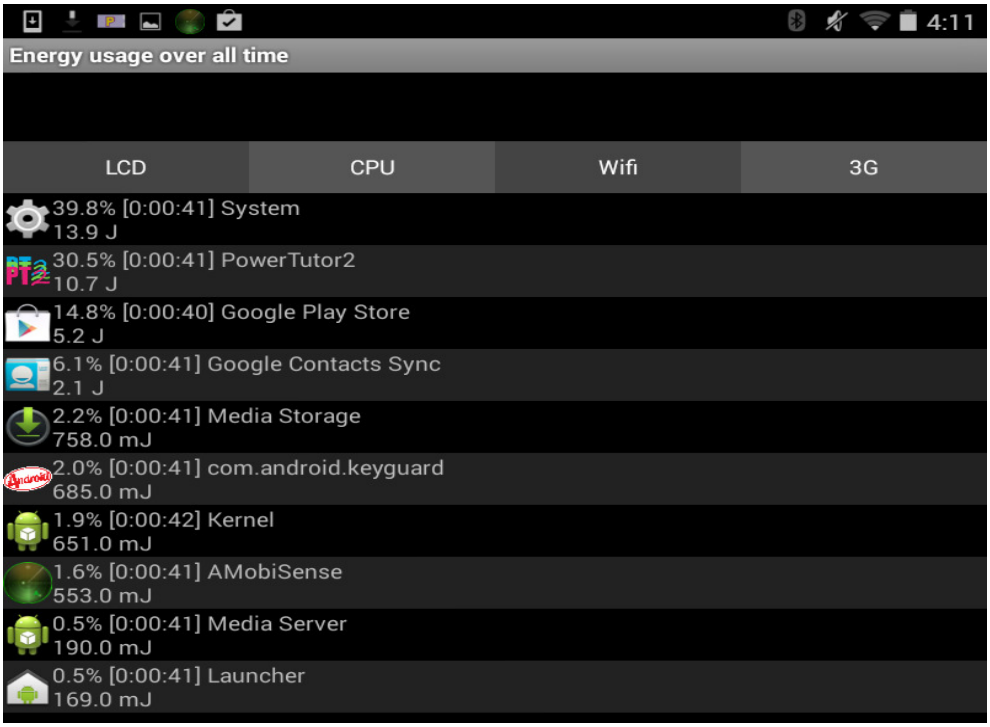

Figure 3.2: Screenshot of CPU utilization and power consumption

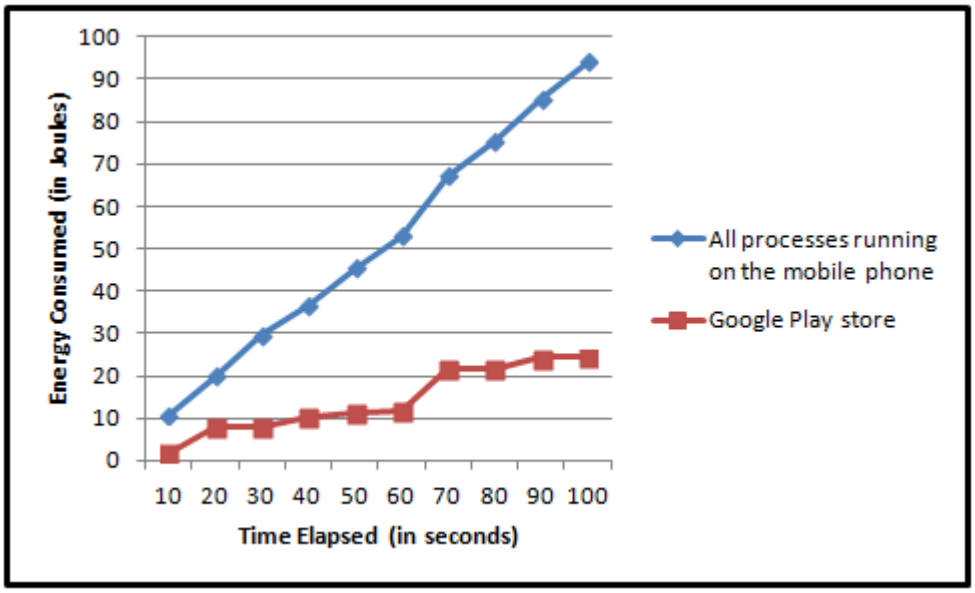

Figure 3.3: Comparison of power consumption

It is clear from Figure 3.3 that if we assume that 100% CPU is available for the Google Play

store to do its operation, the total battery power consumed by the mobile phone will be represented by the line for Google Play store. But since other processes are also running, the total battery power consumed is actually the line for all processes, which has much greater values than with only Google Play store running. So, we have shown that it is necessary to consider that CPU available to our concerned application is much less than 100% and thus any static estimate of the energy consumption will always yield an under-estimate of the total energy consumed.

3.4 Conclusion

In this chapter, we address the problem of energy analysis of a mixed-mode application, using the concept of an usage induced profile automaton. In particular, we wish to answer the question: Given a mobile phone with its current battery condition and an application to run, do we reach a state where all the battery of the mobile phone is exhausted. We believe that our proposal will aid in more realistic energy decisions in comparison to the methods existing in literature. We are currently investigating on application level classification for diverse application types which benefit from our analysis.

Chapter 4

STATEREF: A STATE REFinement technique for REAST

In this chapter, we extend the problem addressed in the previous chapter for a large scale setting. From the previous chapter, we adopt a simple intuitive finite state model for representing the applications running on a user's mobile device, where the applications are represented as nodes of the automaton. The edges between the nodes represent the transitions of the mobile device from one application to another. The applications are considered to be heterogeneous, that is, each application may consume different amount of energy (per second) when run and the average running time of each application may vary. This application finite state machine is created by mining through the work logs of a user's mobile device. We build on the foundation of the application state model proposed in the previous chapter. We monitor a user mobile device for days and weeks and extract the running data of all the applications for that duration of time in the form of a usage log file. This log file is then represented an automaton. In the previous chapter, we have proposed and solved an application scheduling problem by modeling that as a reachability analysis task on the application state model. This forms a key ingredient of our work.

An interesting scenario arises when the usage log file contains a lot of different applications. In this case, the automaton thus created becomes very large to analyze,thereby the naive method suggested in the previous chapter is bound to run into scalability issues. This forms the core motivation of this chapter, whereby, we propose to adopt the notion of iterative refinement on the application state machine to address the scalability issue. The basic technique of refinement we use here is based on the discussion in [33].

Our approach proceeds as follows. We first compute an abstract version of the entire state transition diagram. We do so by combining related applications into single states or clusters. The similarity among different applications can be defined in terms of similar resource requirements, similar skeleton of execution, or any other such parameter. Our

parameter of defining similarity among different applications is the product of the energy dissipation rate and the permanence time the user spends in an application. This makes the state transition diagram more easy to infer with much less number of states which are basically clusters of different applications.

After successfully creating an abstract version of the original state transition diagram, we try to solve our scheduling problem. Since the abstracted version is much smaller, scheduling can be done faster. We analyze the scheduling problem on the abstract state machine. The task of the scheduler is to find if it is feasible to run a desired application on the mobile device without degrading the performance of the mobile device. For this, starting from the state the mobile device is currently in, all the paths need to be checked till the time the execution of the desired application is successfully completed. Thus, the desired criteria for the successful completion of the scheduling problem is the situation where the scheduling algorithm does not find a path which leads to an undesirable state (a state which leads to a decrease in performance of the mobile device). If that is not the case, the scheduler needs to refine the abstraction, which results in a new abstract state machine on which the scheduling algorithm can be applied again. This process needs to be repeated iteratively until the scheduling problem can be solved in the affirmative or we find that it is actually not possible to schedule the application on the given mobile device. Additionally, if the abstract state machine at any iteration expands to the original concrete state machine containing one state per application, and still the scheduling is not possible, then the scheduler declares that it is not safe to schedule the desired application immediately. The ability to automatically and efficiently provide an answer to the scheduling problem is the main crux of this contribution. We provide experimental results to show how our proposal compares with the naive scheduling method proposed in the previous chapter.

4.1 Motivation and Contribution

We begin by describing the concept of the application state automaton that is used in this chapter for solving the scheduling problem through iterative refinement. The application state model is created by monitoring all the states running in a mobile device. This state model is used to answer the question of whether it is possible to schedule an application to run on a particular mobile device at a given time. Application logs of a mobile device are collected and a simple rechability algorithm is run through them to find out: given a mobile device with certain amount of battery remaining, and an application that the user intends to run on the mobile device, is it safe to run the application. In other words, is it possible that while running the application, the entire battery of the mobile device is exhausted and the application is not able to complete. If such a scenario can arise, then the scheduler declares it as unsafe to run the application. The user should run the application only after charging the mobile device properly.

The application usage state model, referred to as *Usage Profile Automaton (UPA)* as de-

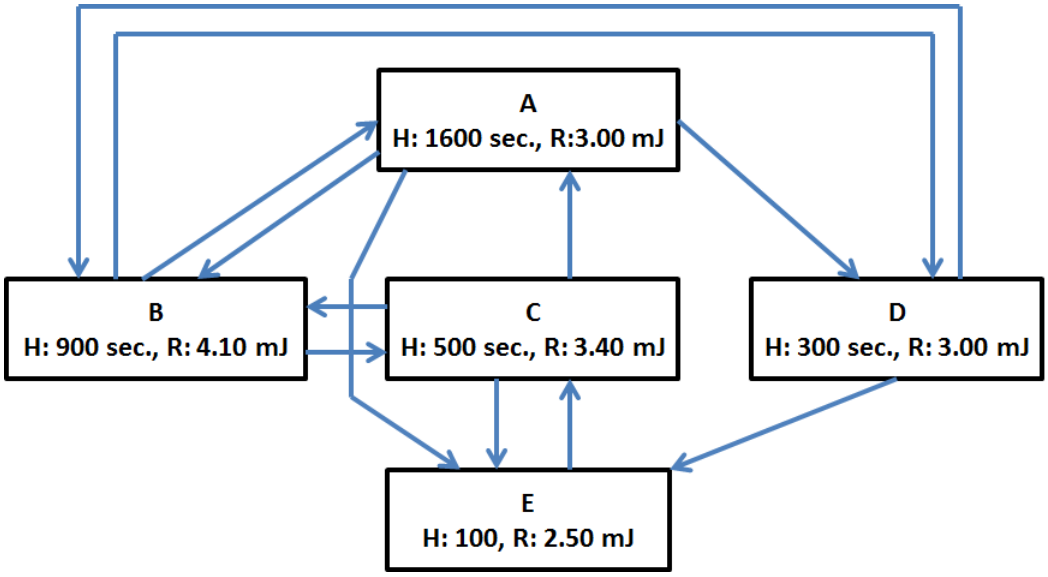

Figure 4.1: Usage Profile Automaton (UPA)

scribed in the previous chapter is created through the logs of the user mobile device. An example state transition automaton is shown in Figure 4.1. The states of UPA are the applications that run on the mobile device during the entire duration of the log. Figure 4.1 has five states labelled A to E. R denotes the amount of energy spent in one second and H is the average permanence time. Now, suppose that while using the mobile device, the user decides to run an application, like a virus scan or a Google Play Store update. It can either be completed quickly or it might take a considerable amount of time for its completion. Consider an application *App* to be run on the mobile device from the current state A. The time required for the complete execution of *App* is 1700 seconds and its rate of energy dissipation is 1 mJ/sec. It is supposed to run on a mobile device having remaining battery energy of 10000 mJ. Suppose the user is currently in state A. The average permanence time of state A is 1600 seconds and its rate of energy dissipation is 3 mJ/sec. After completion of execution of state A, the amount of execution time left for *App* to complete its execution is $1700 - 1600 = 100$ seconds. The remaining mobile battery energy is $10000 - 1 * 1600$ (for *App*) $- 3 * 1600$ (for state A) $= 3600$ mJ. *App* will complete its execution in another 100 seconds. Now, from state A, the user may transit to either state B, state D or state E. Let us first consider state B. State B has an average permanence time of 900 seconds and rate of energy dissipation as 4.10 mJ/sec. In 100 seconds, the amount of battery energy drained of the mobile device is $1 * 100$ mJ (for *App*) $+ 4.10 * 10 = 141$ mJ, after which it still has 3600 mJ $-$ 141 mJ $= 3459$ mJ of battery remaining. This will lead to a successful scheduling of *App* on the mobile device. Next consider the transition from state A to state D. State D has an average permanence time of 300 seconds and rate of energy dissipation as 3 mJ/sec. In 100 seconds, the amount of battery energy drained of the mobile device in state D is $1 * 100$ mJ (for *App*) $+ 3 * 10 = 130$ mJ, after which it still has 3600 mJ $-$ 130 mJ

= 3470 mJ of battery remaining. So, this will also lead to a successful scheduling of *App* on the mobile device. Finally, consider the transition from state A to state E. State E has an average permanence time of 100 seconds and rate of energy dissipation as 2.50 mJ/sec. In 10 seconds, the amount of battery energy drained of the mobile device in state E is $1 * 100$ mJ (for *App*) + $2.50 * 10$ = 125 mJ, after which it still has 3600 mJ − 125 mJ = 3475 mJ of battery remaining, which will lead to the successful scheduling of *App* on the mobile device. Hence the scheduling algorithm will give success as its output since all the paths resulted in the completion of the execution of *App* without the mobile device exhausting its entire battery energy. Thus it is necessary to know whether it will be successfully completed or not, if the application is scheduled to start execution from the current state. A simple reachability analysis from the current state can reveal the answer. If there exists a path on which the residual battery left after the application execution reaches an undesirable value, it is not advisable to schedule the application from the current state. Intuitively, the user may actually end up selecting any of the outgoing paths from the current state during actual execution. Thus our conservative analysis is of immense value. The application state model helps us to analyze these questions statically.

The motivation for this scheduling problem has been from the work done by Hao et. al. [32]. In [32], the authors have proposed a novel approach for calculating the energy consumption of a mobile application through program analysis. They have suggested a tool and have extracted as its output, the code-level estimation of the energy that an application A will consume at run-time, through analysis of the implementation of the application.

The analysis done by the algorithm proposed in the previous chapter is simple when the number of applications recorded by the log file is few in number. However, this is not always the case. Imagine a scenario in which the log file has the data of a few days which contains a large number of applications, for which a naive explicit state representation and traversal may not scale to handle the task of suggesting if all suitable paths are safe. Initially to traverse all the possible paths of the automaton just to find if there exists that one path which is unsafe is unrealistic for a huge group of applications.

Consider a Usage Profile Automaton which has around 1000 states. When the scheduler wishes to start an application (virus scan or Google Play store update), suppose that the user is in state A. From state A, the user has many options. In case of a complete graph, the user has 1000 options. After the user transits to another state, again he/she has another 1000 options to choose from. If the application runs for a long duration of time, the total number of different paths to be considered will simply blow up, and the result will take a very long time to compute.

It may be noted here that our main task is only to suggest if it is safe to run the application. To be able to handle a large state transition automaton, we propose here, a cluster-based scheduling analysis approach. Each cluster will has a subset of applications from the mobile device's log and each application belongs to one and only one cluster. The clusters can be created using standard clustering methods. Now the scheduling algorithm boils down to a

rechability analysis on a clustered state machine with possibly multiple iterations to answer the scheduling question. We explain below the philosophy behind our proposal through a simple example.

Cluster based analysis

The objective of the cluster construction step is to create application clusters such that it is more amenable to schedulability analysis. To do so, we merge different application states with similar parameters into clusters. Each cluster has a subset of applications from the mobile device's log and each application is in one and only one cluster. If two application states which fall within the same cluster after clustering had a transition earlier from one to the other, the edge will be represented as a self loop on the corresponding cluster. Each time the application takes one iteration of the self loop, a quantum of permanence time is elapsed. In case there is a transition from an application from cluster C_1 to an application in cluster C_2, there will be a transition from cluster C_1 to cluster C_2.

The creation of clusters is done by assigning each application to a cluster through some cluster creation technique. Here, we use both the parameters used to represent the states of the Usage Profile Automaton, the rate of energy dissipation and the average permanence time. For each application within a cluster, we calculate the product of its rate of energy dissipation and its average permanence time. We call this product as *state energy consumption*. Then we use clustering methods to create clusters keeping the state energy consumption as the sole parameter. The value of the rate of energy consumption parameter of a cluster is the highest rate of energy consumption value among all the application states belong to that cluster. Similarly, the value of the average permanence time parameter of a cluster is assumed to be the longest average permanence time value among all the applications in that cluster. The reason for taking the maximum values for both, the rate of energy consumption and the average permanence time, from all the applications of a cluster is to provide a conservative over-estimate of these values. In our work, we take the product of these two parameters for each cluster, which we have coined as *state energy consumption*. The amount of energy consumed in a particular state is the energy dissipated per second multiplied by the number of seconds the user remained in that state, which is nothing but the state energy consumption of that state. So after the formation of the clusters, the value of the state energy consumption parameter of a cluster will be greater than or equal to the value of the state energy consumption parameter of any of the applications present in that cluster (the equivalent scenario will arise when in a cluster, the same application had the maximum value for both, the rate of energy consumption and the average permanence time). This leads to the conservative over-estimate since no more than the value denoted by the state energy consumption can be spent at a time in a cluster, that is, the user can not spend more time in a cluster of applications than the average permanence time of that cluster, and the rate of energy dissipation during that duration can not exceed the rate of energy dissipation of the same cluster.

We now analyze the application scheduling problem from the perspective of the clusters. After we have created the clusters using certain cluster formation algorithm, we can adopt a simple reachability method from the current cluster (the current state of the original concrete usage profile automaton gets mapped to some cluster which therefore becomes the current cluster to start from). We consider the clusters as states of the state transition automaton and try to calculate if a path can be traversed such that we end up reaching a state where the entire battery energy of the mobile device is fully consumed. We explain the situations that may arise below by a couple of intuitive examples. Let us consider the case that the schedulability analysis for an application *App* starts at a particular state A of the concrete usage profile automaton which maps to the state S of the clustered automaton. The *App* has an energy requirement of E J. The state parameters of the clustered usage profile automaton have been assigned as discussed above. There are two cases that need to be considered here.

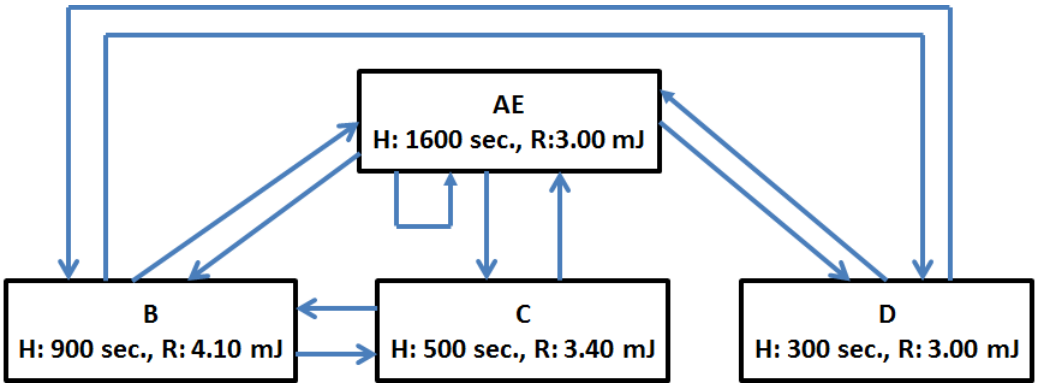

Figure 4.2: Case: Safe to schedule

Case 1: If all paths from state S in the clustered machine result in the completion of the application *App* without the battery getting depleted, then we can declare that it is safe to run the application on the original unclustered usage profile automaton as well. This is possible because we took the highest energy consumption parameter and the longest average permanence time parameter from all the applications present in each cluster. So, we considered the worst case energy consumption that is possible, since an application in a cluster will have its state energy consumption parameter less than or equal to the state energy consumption parameter of the cluster. For an example depicting cluster creation, consider the clustering shown in Figure 4.2. Application states A and E have been clustered together to create the cluster AE (also see Figure 4.1) which has the highest rate of energy dissipation and the longest average permanence time among the two application states. There are only four states now on which the schedulability analysis is to be performed and we indeed have no unsafe paths here.

Case 2: If the schedulability analysis uncovers a path on which the entire battery of the mobile device is consumed by running *App*, then we can have three scenarios.

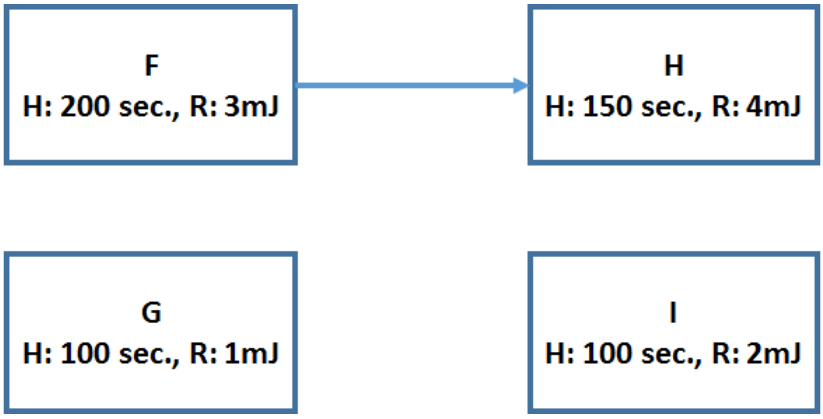

Figure 4.3: A state transition automaton with an edge from state F to state H

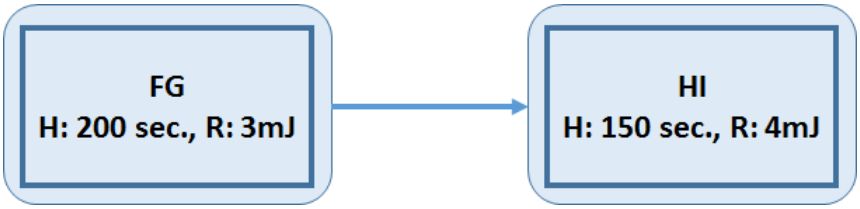

Figure 4.4: Clustered state transition automaton

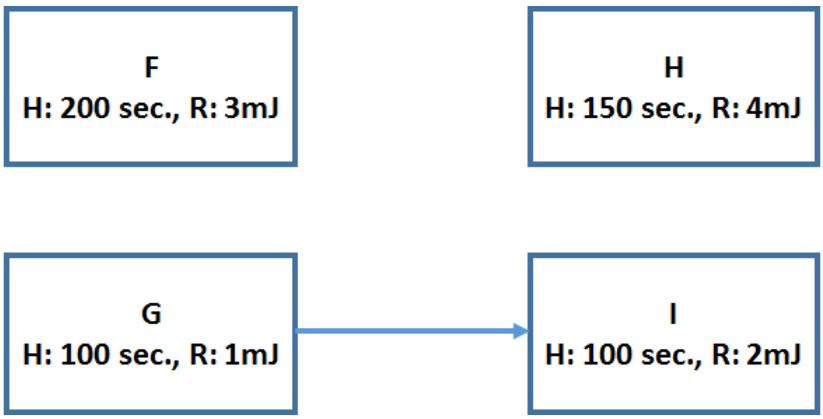

Figure 4.5: A state transition automaton with an edge from state G to state I

- In the first scenario, the failure path uncovered by the scheduling algorithm on the clustered abstract state machine is actually a valid path in the actual unclustered state transition automaton. Also, for all the clusters, the rate of energy consumption and the average permanence time have been derived from a single application of each cluster. In this case, the failure reported is actually a true one. For example, consider the state transition automaton shown in Figure 4.3. After clustering, the state transition automaton is shown in Figure 4.4. In this scenario, the path from FG

to HI is a valid path since both the parameters of FG and HI have been contributed by states F and H respectively, which themselves have a path in between them.

- In this case, the failure path uncovered by the schedulability analysis algorithm is *not* a valid path on the original unclustered automaton. Again, for all the clusters, the rate of energy consumption and the average permanence time have been derived from a single application of each cluster. Hence the schedulability answer is not correct. The failure result which comes out of the schedulability analysis is a side-effect of the clustering we had done and not a true failure because of the transitions and the self loops. The transitions may actually lead the schedulability analysis to infeasible paths which are not possible on the unclustered state machine and thus gives a false failure result. Consider the state transition automaton shown in Figure 4.5 where there is a path from state G to state I. After clustering, the state transition automaton will be same as that shown in Figure 4.4. In this scenario, the path from FG to HI is *not* a valid path since both the parameters of FG and HI have been contributed by states F and H respectively, which do not have a path in between them in Figure 4.5. This is where we need to refine the cluster and possibly come up with a different abstract UPA on which the schedulability answer can be collectively deduced. Refinement is necessary since we have taken a conservative over-estimate of the *state energy consumption* parameter of every cluster, thus resulting in the exhaustion of the battery energy of the mobile device. So a refinement is done with the hope that such set of clusters might be found which will result in a safe state for the application *App* to run.
- In the final scenario, it may happen that for at least one cluster, both of its parameters have been derived from two different applications within that cluster. Here also, the next step will be refining the state diagram.

To understand which of the above scenarios are applicable, we need to analyze the unclustered state machine and proceed accordingly. In the section below, we formalize our approach of clustering, refinement and schedulability analysis.

4.2 The overall methodology

When there are a large number of applications whose states need to be analyzed in the state diagram, the state diagram might blow up. So, different clusters with energy threshold can be created where each state is assigned to a particular cluster based on some energy threshold. The main idea is as follows:

1. Divide all the application states into a number of clusters (C), using an algorithm which will be elaborated shortly, based on the product of rate of energy consumption and average permanence time of individual applications (the state energy consumption parameter).

2. Let the highest value of the rate of energy consumption among all the applications in a cluster $c \in C$ be R_i (i is the application having highest rate of energy consumed in cluster c), and the longest average permanence time of an application in the same cluster be H_j (j is an application in cluster c where i may or may not be equal to j). Assign the rate of energy consumed and the average permanence time parameters of the cluster c as R_i and H_j respectively. Then the *state energy consumption* parameter of cluster c will be $R_i * H_j$.

3. The application states present in the unclustered state diagram are now replaced by states (C) representing few different clusters of application states. The new state diagram is created as follows:

   - We create the C micro clusters as the C states of the new state automaton.
   - We add self loops on clusters corresponding to the transitions (if any) between two states inside the same cluster.
   - For each transition between two states belonging to two different clusters, we create an edge from the cluster to which the source state belongs to the cluster to which the destination state belongs in the new state diagram, if not already drawn.
   - The value of the rate of energy consumption parameter of a cluster is the highest value of the corresponding parameter among all the applications states in that particular cluster. Also, the value of the average permanence time parameter of a cluster is the longest value of the corresponding parameter among all the applications in that particular cluster.

4. The schedulability analysis starts with the cluster which holds the state currently running on the mobile device. Then the normal paths are followed to see if the desired application that the user wants to run reaches completion before the entire battery of the mobile device is exhausted on any path. The following situations can arise as discussed earlier.

   - We do not find such a path. We declare that it is safe to schedule the desired application.
   - Such a path could be found. In other words, before the application could complete, the mobile device will exhaust its entire battery power. In this case, we need to examine the unclustered state machine and proceed accordingly. If the path is a valid one, we conclude that scheduling is not possible. If the path is invalid, and not present in the unclustered machine, we need a refinement step. In this case, we create new clusters by replacing/dividing old clusters. The creation of new clusters is done by the *Similar Duration Job Abstraction algorithm* which is a refinement algorithm. Based on the algorithm we decide the number of clusters to be formed and what applications need to be assigned to which cluster. We then execute steps 2 to 4 iteratively. If we have broken the clusters to a depth such that there are no more clusters and each application is represented as

a single state, and we still have not have declared that it is safe to schedule the desired application, then we declare that it is unsafe to schedule it. Hence the user should charge the mobile device properly before scheduling the application.

4.3 The iterative refinement framework

We discuss about the formal model that we used in our work. The Usage Profile Automaton is same as that used in the previous chapter minus the CPU availability parameter.

- S, denoting the set of application states $\bigcup$ the (idle, switched off) states.
- E is the transition relation between application states.
- H: S $\rightarrow \mathbb{R}$ represents the average permanence value defined as H(s_i) = t_i, where $s_i \in$ S, $t_i \in \mathbb{R}$.
- R: S $\rightarrow \mathbb{R}$, denoting the rate of energy consumption at a state.

Each application that the user runs on his device appears as a state in S as collected from the logs. For each transition observed between application states in the logs, we add a transition to E. The rate of energy dissipation also includes the contribution of the routine system processes. The scheduling problem considered here is as follows. Given a state s $\in$ S and remaining battery level l, is it safe to run an application A with energy requirement E_A and the rate of energy dissipation R_A given that the user has already spent some time in the state s, and may transit to any other state after the average permanence time for that state is over. To answer the question above, we need to check if the zero battery state is reached by the time this application finishes. To do so, we first create the abstract clustered UPA as described below.

The clustering problem

Formally, the parameters of the clustered UPA are as follows:

- C, denoting the set of clusters, where each cluster contains a set of applications states $\bigcup$ the (idle, switched off) states. Each application state is present in exactly one cluster.
- E is the set of edges. If two application states belonging to the same cluster have an edge in between them, then add a self loop to that cluster. If an edge is present between two application states belonging to two different clusters, then there is an equivalent edge in between the two clusters.

- H now represents the average permanence value.
- R is the highest rate of energy consumption value among all the applications present in that cluster.

The clusters are created using the α-Similar Duration Job Abstraction algorithm as described below.

Similar Duration Job Abstraction

An α-similar duration job abstraction is a partition of the applications into many clusters in the state diagram such that the application with the highest value of the state energy consumption parameter is at most α times the shortest one in every cluster. Initially, we will set the value of the α parameter as 3. There is some pre-processing required to sort all the applications in non-decreasing order and store them.

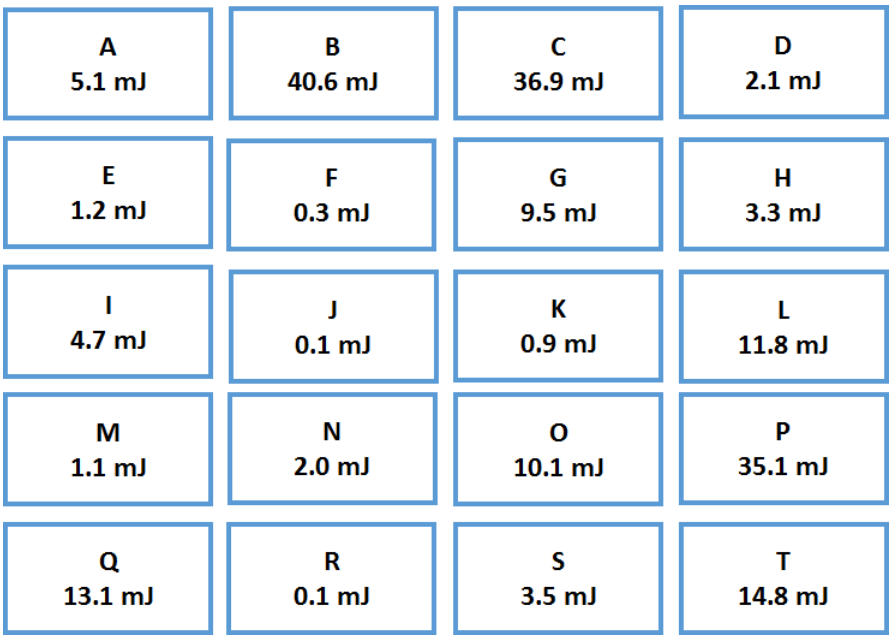

Figure 4.6: Unsorted set of applications

The basic algorithm using similar duration job refinement is as given in Algorithm 2 (initially, the value of α is 3). Our algorithm for refining using similar duration job abstraction is given in Algorithm 3. Algorithm 3 is the similar duration job abstraction refinement algorithm. If we found a path in Algorithm 2 which leads to a state having zero mobile energy left, then Algorithm 3 will be executed. In Algorithm 3, if in each cluster, the value of the rate of energy dissipation and the average permanence time of the cluster was taken

Algorithm 2 Similar Duration Job Abstraction algorithm

Take as input all the applications sorted in non-decreasing order based on their state energy consumption
Assign the first application as the head of the first cluster
With the current value of α, add applications in the sorted order to the first cluster
After encountering an application A which falls outside the range of α times the first application, assign application A as the head of a new cluster
Continue this process till all the applications have been covered
Apply the scheduling algorithm
if Success **then**
    Declare that it is safe to schedule the desired task
    EXIT with success
else
    if both the rate of energy dissipation and the average permanence time parameters of each cluster is same as that of any individual application within that cluster **then**
        Examine the path for validity.
        if path is found to be valid **then**
            EXIT with failure
        else
            Run Algorithm 3
        end if
    else
        Run Algorithm 3
    end if
end if

Algorithm 3 Refinement using Similar Duration Job Abstraction

Take the clusters as input
if all the clusters are individual applications **then**
    Declare that it is not safe to schedule the task
    EXIT with failure
end if
$energy\_difference$ = Energy requirement of the task − Energy that the task spent in the previous step
$energy\_percent = energy\_difference$ / Energy requirement of the task $*$ 100
Round up $energy\_percent$ to single decimal digit
$\alpha = \alpha - energy\_percent$
Redo Algorithm 2 using the new value of α

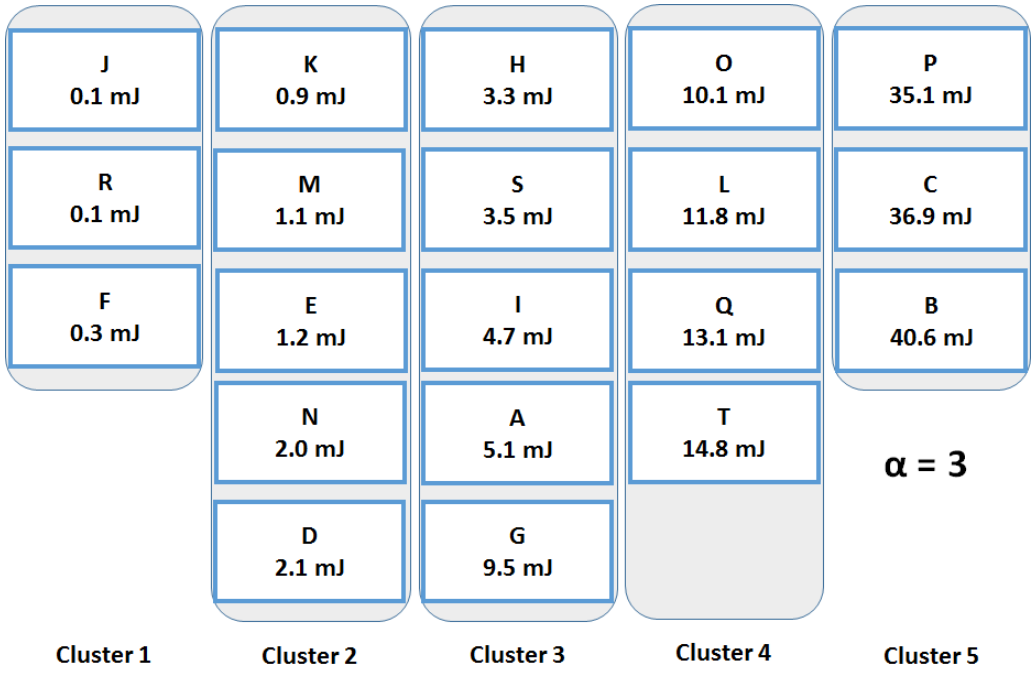

Figure 4.7: Clusters of applications with $\alpha = 3$

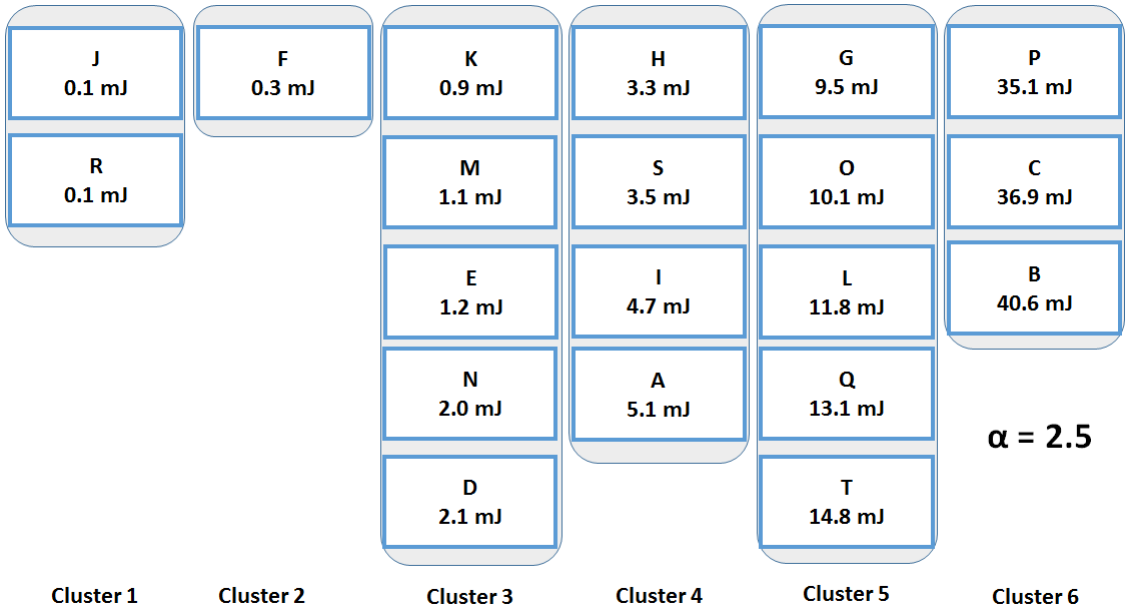

Figure 4.8: Clusters refined with $\alpha = 2.5$

from a single application for each cluster (Line 2), or if all applications are individually

represented as states and no more cluster exists (Line 6), then no safe path exists. Otherwise, we calculate the percentage of energy that the desired task still needs to complete its operation, round off the value to single digit of decimal, and subtract it from the current α value to create the new α value. Then, Algorithm 2 is called again.

Let us consider a hypothetical example. Figure 4.6 shows an unsorted set of 20 applications, labelled from A to T. The numbers with each applications represent the state energy consumption parameter, that is, the product of the rate of energy dissipation and the average permanence time for that state. We assume that these states are part of a state transition automaton having edges in between them which depict the transitions between any two applications. By applying the clustering step of Algorithm 2, we will get a structure as shown in Figure 4.7. Basically, all the applications are sorted based on their state energy consumption values in non-decreasing order and then clustered by taking $\alpha = 3$. Then suppose that a path is found which leads to a state where the mobile device's battery will be completely exhausted after only 50% of the desired task's energy has been spent. So, the amount of energy of the desired task that was left to be spent is 50% or 0.5 times the total energy required by the desired task. Thus, the new value of the parameter α is $3 - 0.5 = 2.5$. We then perform the refinement algorithm, that is, Algorithm 3. The resulting output is shown in Figure 4.8.

4.4 Experimentation results

We conducted a number of random simulations for testing the scalability of the scheduling algorithm without clustering, and for the scheduling algorithm with clustering and refinement (using similar duration job abstraction algorithm). The usage logs were collected from real usage profiles by our android application. The results are summarized in the graph plotted in Figure 4.9. For each of these, we have considered that the desired task that the user wants to run requires 100 mJ of energy to complete its execution. The rate of energy dissipation for the task is 0.1 mJ per second., and the remaining battery energy of the mobile device is 10000 J. We considered fifteen different cases of the usage profile automaton with varying number of states between 600 and 2000, which we have used as the horizontal axis of the graph shown in Figure 4.9. The time taken is represented on the vertical axis. The upper curve represents the values obtained when a simple scheduling algorithm is performed without clustering, whereas the lower curve represents the values obtained by our proposed algorithm which supports clustering of applications and the refinement of those clusters.

As it can be seen, there is a significant time difference in the performance between the algorithms. Also, the increase in the time consumed in algorithm supporting clustering (and refinement) is almost uniform (lower curve) as the number of application states in-

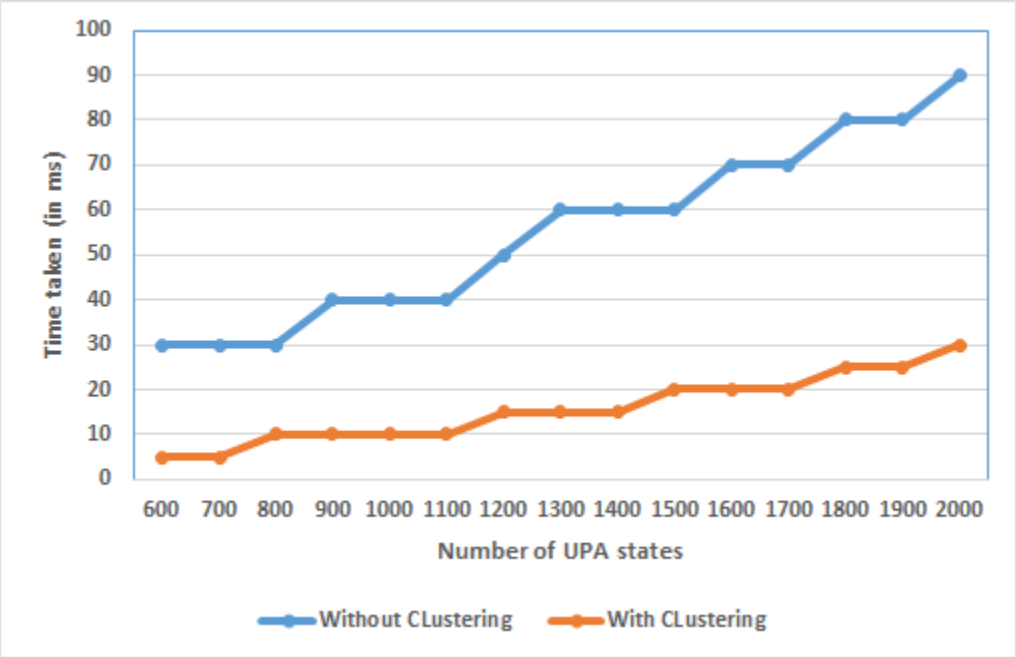

Figure 4.9: Comparison of clustering algorithms

crease, whereas for the simple scheduling algorithm which does not support clustering (upper curve), the increase in time consumed is more as the number of application states increase. This justifies the fact that clustering and refinement solves the scheduling problem faster than simple scheduling algorithms in the average case. However in the worst case it may so happen that the original unclustered state machine does not really contain a path which is safe, and therefore our clustering and refinement algorithm runs through multiple iterations and finally reaches a single state unclustered UPA and terminates with a negative answer. In this process, it consumes more time than what a naive scheduling algorithm without clustering would have taken. However such situations were rare and we could not find any in our random experiments.

4.5 Conclusion

The application schedulabilitiy analysis problem on a mobile device with a significant number of application states in a usage log, can consume a lot of time if all the applications are explicitly represented. This chapter adopts a clustering based iterative refinement approach to solve the schedulability analysis problem. Preliminary experiments show that our intuitions are well justified. We are currently working on extending our analysis to a framework and an android application which can be widely used.

Chapter 5

VARES: A Variation Aware Residual Energy Scheduler for mobile cloud computing

Offloading in Mobile Cloud Computing (MCC) is a key way of mitigating resource constraints of mobile devices like smartphones. In offloading, some of the tasks within a mobile application are migrated and executed on resource-rich cloud systems. Intelligent selective migration of energy or computation-intensive tasks have shown significant energy and time savings

In many mobile applications, such as cloud gaming and video streaming, the quality of service offered to the user can vary depending on the availability of resources. For such applications, proper adaptation of quality of service *and* scheduling of tasks on the mobile device or cloud is essential. This allows such applications to optimize the quality of service given to the user while satisfying the available energy budget.

Let us take a gaming application as an example. In gaming applications, the quality of service is measured using the amount of detail shown in the application. In such a scenario, increasing the amount of detail makes the game more realistic. However, this increases the amount of data transfer through the network, thus increasing the energy consumption. Thus, one objective of the MCC system is to ensure that the best possible quality of service is provided while adhering to the energy budget limitation of the user.

In this chapter, we first develop two different application models of the MCC system to formulate respective scheduling problems. One application model is a linear model to solve linear work flows and another a concurrent model for concurrent work flows. With the help of these models, we develop the respective linear and concurrent algorithms to optimize the quality of service under energy budget constraint. We also show that our algorithm

provides the optimal solution in polynomial time.

We divide an application into several tasks such that each task has various variants, each variant having different QoS parameters. The crux of our model development is to divide each task into variants and to represent each variant distinctively, as an individual node in a graph depicting the entire application. This results in an application graph where each task of the application is broken down such that each variant of each task is represented as a node in the graph. We then apply a scheduling algorithm to find different *paths* which symbolize different possible schedules of the graph. Finally we choose one schedule from the various schedules which satisfies the energy budget, also provides the best number of variants for that energy budget.

Our contributions may be summarized as follows:

- We develop two formal models, a linear one and a concurrent one, to optimize the quality of service under energy budget constraint.
- We propose polynomial algorithms to solve the respective problems.
- We analyze our algorithm mathematically and through trace-driven simulation to show it provides an optimal solution, and is effective in practice.

5.1 Problem Description

Our problem involves scheduling an application to run. The application consists of various micro tasks which can be scheduled in some topological order (based on certain dependencies among the tasks). Each task of an application can have different number of variants. Each of these variants can be executed locally on the mobile device. These variants can be numbered from 1 to n where 1 is the best variant of a task that can be executed locally and n is the worst variant of the same task that can be executed locally. Apart from the option of executing each task locally through one of its variant, a task can also be offloaded to the cloud for its execution. When on the cloud, only the best variant of a task will be executed. We number the cloud variant of a task as 0.

The execution of an application on the cloud does not exhaust any mobile device energy. But the trade-off is the data transmission energy and the data receiving energy (with respect to the concerned application), which need to be accounted for. In the mobile device, there is no transmission/receiving energy for the data but there is energy exhausted of the mobile device to execute the application.

Since we are assuming that the tasks of an application have certain topological ordering, this order can be followed to execute the tasks. The problem of scheduling a task of an application involves deciding whether to schedule a task on the cloud or locally on the mobile

device, and if a decision is made to schedule the task locally, then which variant of the task is to be executed. Say that variant v of task x has been scheduled to execute locally. Since the task is a local task, it will consume the mobile device's energy and this energy consumption needs to be taken into account while calculating the total energy consumed by the schedule. Also, if the previous task or the next task or both (in topological ordering) were scheduled to execute on the cloud (variant 0), then there will be some energy consumption in downloading the input if the previous task was on the cloud or/and offloading the output if the next task is scheduled on the cloud. If a task which immediately precedes or succeeds the current task in topological ordering is scheduled locally, then there will be no energy consumed to transfer the input to the current task or to supply the output of the current task. The transfer energy will also be non-zero if the previous or the next task is to be executed on the local device and the current task is scheduled to execute on the cloud. Finally, the amount of energy consumed by the mobile device in performing the cloud variant of a task is zero since the mobile device will play no part in its execution.

The quality of a variant is described by the QoS parameter of that variant. So in our case, for a task x, the QoS parameter of variant 0 and variant 1 will be same and also the highest among other variants. Let the energy consumed by the best variant of a task x be e. The the energy consumed by the second variant of task x will be e/α, where α is a constant. The energy consumed by the third variant of task x will be e/α^2. Similarly, the energy consumed by the i^{th} variant will be $e/\alpha^{(i-1)}$.

5.1.1 Mobile Cloud Computing System

The entire application containing various tasks is represented in the form of a Directed Acyclic Graph (DAG). Each node in the graph is represented as a task of the application. There is an edge from a task x to another task y only if the start of task y is dependent on the completion of task x. Each node has an energy parameter associated with it called the node energy, which denotes the amount of energy that the node will consume for its execution. The cloud variants of all the tasks have the node energy's value as zero. There is also energy consumption parameters associated with each edge. We will call this parameter as edge energy. For local mobile device to local mobile device transition and cloud to cloud transition, the edge energy is zero. It is assumed that the first and the last tasks of the application are native tasks and have a single variant.

5.1.2 Linear Application Model

A linear work flow of an application can be seen in Figure 5.1. For the formation of the linear application model, we will break each non-native task of the graph in Figure 5.1 and expand it so that each variants of each task is a node in the new graph. The linear application model for Figure 5.1 is shown in Figure 5.2. Each level of nodes in Figure 5.2

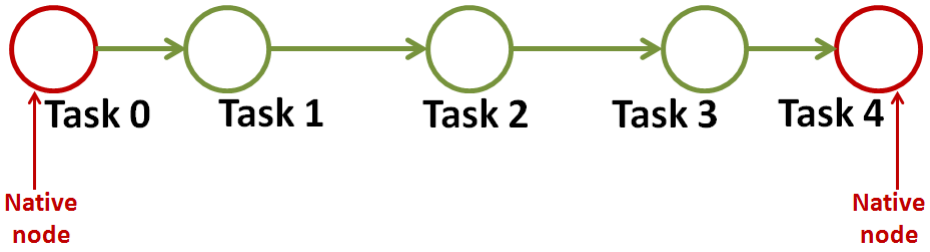

Figure 5.1: Linear work flow of an application

represent a single task and each task is divided into three variants, represented as nodes. The node energy is written inside each node and the edges are represented by the edge energies. The nodes are labelled from 0 to 10. Nodes 0 and 10 are native tasks. Node 1 represents the cloud variant of second task, node 2 represents the first variant of the second task that can be executed on the local device and node 3 represents the second variant of the same task. Thus, all blue nodes represent the cloud variants, all grey nodes variants represent the first local variant and all brown nodes represent the second local variant of their respective tasks. The transitions between the nodes are same as described above.

Given this model, the scheduler will suggest a path from the start node (node 0) to the destination node (node 10) such that exactly one variant of each task is chosen to be executed. The constraint of choosing exactly one variant from a task is realized in Figure 5.2 since there is no edge between any two nodes belonging to the same layer.

5.1.3 Concurrent Application Model

The concurrent model faces the problem of of having concurrent nodes, which results in a non-linear model. A concurrent work flow can be seen in Figure 5.3. Figure 5.3 can no longer be represented as a layered graph since node 0 has two outgoing edges, hence the structure of the graph is no longer linear. To solve this problem, a topological sort of all the nodes in the work flow can be done. A topological sort of the work flow in Figure 5.3 is shown in Figure 5.4. Now, the graph in Figure 5.4 is linear and the scheduler can now suggest a path from start node to destination node by again representing the graph in Figure 5.4 in its expanded form as shown in Figure 5.5. An important point to note here is that there might be some edges lost and/or some new edges added in the topological ordering of nodes. For example, in Figure 5.4, edge from node 1 to node 3 has been added which was not present in the original work flow in Figure 5.3. Also, edge from node 1 to node 2 is absent in Figure 5.4. The energies of the absent edges need to be accounted for by the scheduler, and those edges which have been newly added can be simply represented by an edge having edge energy as 0.

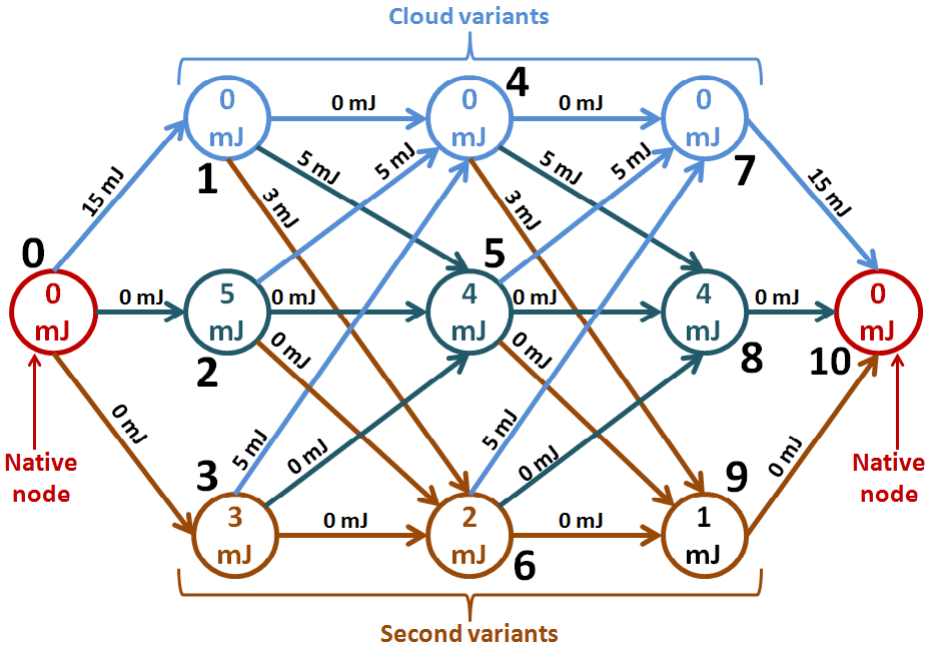

Figure 5.2: Linear Application Model

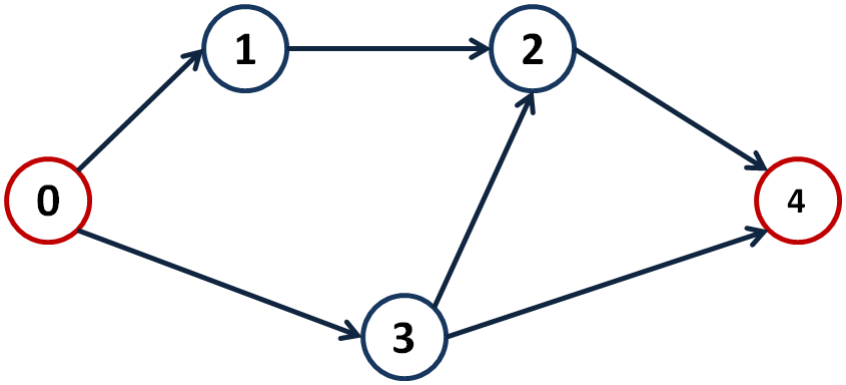

Figure 5.3: Concurrent work flow of an application

Figure 5.4: Topographical ordering of the concurrent work flow

5.1.4 Quality of Service Optimization Problem

Given any of the above models, our aim is to optimize overall QoS of the entire application. In other words, the scheduler needs to choose a path with as good QoS parameters as

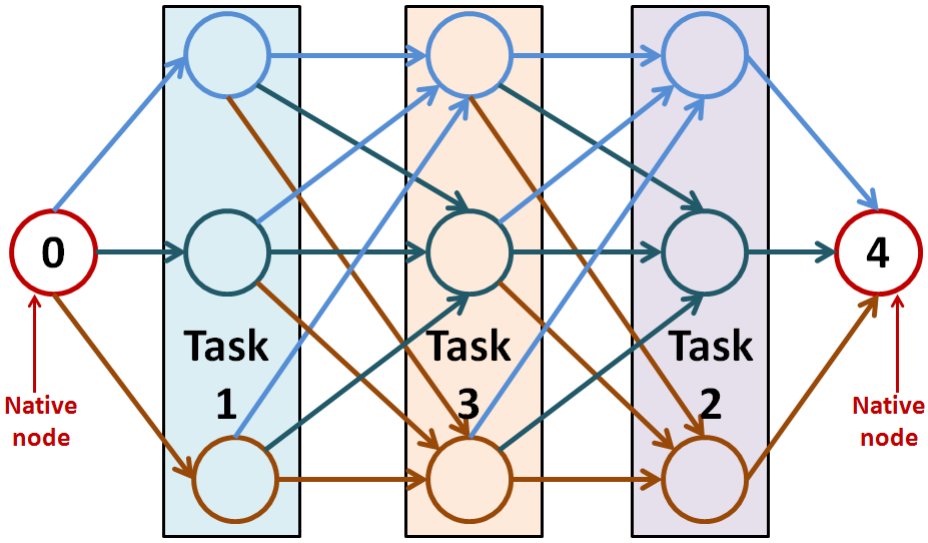

Figure 5.5: Model of an concurrent application after topological ordering

possible for each and every task of the application. The constraint is that the energy consumed by scheduling the chosen set of variants should not cross the energy threshold given by the application user. If it exceeds the energy threshold barrier, a path with inferior variants have to be chosen. Now assume the cloud variant and the best (first) variant that can be executed locally on the mobile device having weightage of 1 in the QoS calculation of the entire application. We will call this weightage of a variant as the index of that variant. The rest of the local variants have indices equal to their variant number, that is, starting from 2 to n, where n is the total number of variants of a task. Let there be total N tasks in an application A, and the variant chosen by task x_i for scheduling has its index value equal to $index_i$. Also, let the total energy consumed by the entire application A be E. Mathematically, the optimization problem can be represented as

$$\forall x \in A, maximize \sum_{i=1}^{N} index_i \ s.t. \ E \leq energy\ threshold$$

5.2 Algorithm for Linear Work flows

The sequential algorithm to solve the problem scenario is given by Algorithm 4. The parameters used in Algorithm 4 are explained below.

- *node_e[i]*: The amount of energy to be consumed by node i. For cloud variant, it's value is zero.
- *I_sum[i][j]*: The sum of indices of a path traversed from source node till node i. The

entry number of that path is denoted by j.

- *E_sum[i][j]*: The sum of the total energy consumed while traversing a path from source node till node i. The entry number of that path is denoted by j. The sum includes the node energy consumed as well as the transition energy consumed while traversing the path j.
- *path[i][j]*: The actual paths which are traversed from source to node i is stored in this array, numbered by j for the node i. For cloud variant, 0 is appended to the path. For other mobile variants, the variant number starting from 1 to n is appended to the path, where n is the total number of variants.
- *node_n*: Represents the node number of the node currently being explored.
- *entries[i]*: The total number of entries that have been stored corresponding to node i. Each of these entries has a value in the arrays *index_sum*, *E_sum* and *path* corresponding to node i.
- *parents of node_n*: All the nodes which have a transition edge to the *current_node*.
- *edge_e[i][j]* : Energy consumed during the transition from node i to node j.
- *E_threshold*: Entered by the user. Denotes the maximum amount of energy that can be exhausted to complete the entire task.

Explanation of Algorithm 4

Algorithm 4 works as follows. Each node maintains some particular number of entries. For each corresponding entry, there are values stored in *I_sum*, *E_sum* and *path* arrays for that node. The algorithm in Figure 5.2 starts with the first node, node 0. From node 0, it explores all possible paths. It can go to three nodes, nodes 1, 2 and 3, which are basically a cloud variant and two local variants of a single task. Then the control moves on to node 1. The parent of node 1 is node 0. The sum of indices till node 1 is 2 (1 for node 0 + 1 for node 1). Since currently there are no other entries for node 1, a new entry is created storing the sum of indices in array *I_sum*. In the corresponding entry in the *E_sum* entry for node 1, the sum of energies of node 0, the edge energy from node 0 to node 1 and the energy of node 1 is stored. The path traversed till node 1 is "10" (1 for node 0 and 0 for the cloud variant, that is, node 1). This value is stored in the corresponding entry of *path* array for node 1. For node 2 and node 3, a similar procedure is carried out. The entries for node $2's$ *I_sum* and *path* arrays will be 2 and "11" respectively. Similarly, the entries for node $3's$ *I_sum* and *path* arrays will be 3 and "12" respectively.

Now we will consider what happens at node 4. There are three parent nodes of node 4 namely, nodes 1, 2 and 3. First we will consider the parent node 1. The sum of indices

Algorithm 4 Algorithm for solving Linear sequence of processes

while *all the parents of node_n have been explored* **do**
  for $i = 1$ *to entries*[*parent*] **do**
    $E\_sum \leftarrow E\_sum[parent][i] + edge\_e[parent][node\_n] + node\_e[node\_n]$
    $I\_sum \leftarrow I\_sum[parent][i] + index[node\_n]$
    Save the path in the *path* array
    if *I_sum* already exists in an entry of array *I_sum* **then**
      keep the lower energy sum in the corresponding entry of the array *E_sum* and change the entry in the *path* array accordingly
    end if
  end for
  Explore the next parent
end while
Sort the array *I_sum* in ascending order and change the entries of *E_sum* and *path* accordingly
for $i \leftarrow 1$ *to entries*[*last_node*] **do**
  if *E_sum of that entry of the last_node is* $\leq$ *E_threshold* **then**
    print the entry's path, E_sum, I_sum
    EXIT
  end if
end for
print no path exists with energy less than the given *E_threshold*

of the first entry of node 4 will be 3 (2 from parent node and 1 for node 4), and this value will be stored as the first entry of the I_sum array for node 4. The energy stored in E_sum array of node 1 will be summed up with the edge energy from node 1 to node 4 and the node energy of node 4, and stored in the corresponding entry of the E_sum array for node 4. The value of the *path* array's first entry for node 4 will be "100" (10 from node 1 and 0 for the cloud variant, that is node 4).

Now we will go on to the next parent of node 4, that is, node 2. The sum of indices will be 3 (2 from parent node and 1 for node 4). But the value 3 is already stored as the first entry of the I_sum array for node 4. So the sum of energies of node $2's$ E_sum array, the edge energy of the edge from node 2 to node 4 and the node energy of node 4 will be added. This energy sum is smaller that the energy sum stored as the first entry of the E_sum array for node 4. So the previous stored energy sum value is replaced with the new smaller value in the first entry of the array E_sum for node 4. The *path* array for the first entry of node 4 is updated to "110" (11 from node 2 and 0 for the cloud variant, that is node 4). We keep only a single entry for a particular index value sum for a node. The proof of keeping the entry with smaller energy sum among the entries having the same index sum is given later in this section.

The next parent of node 4 is node 3. We find that the sum of indices is 4 (3 from node 3 + 1 for node 4). This value does not exist in the array I_sum of node 4. So a new entry is created in array I_sum in which the value 4 is stored. New entries are also created for the arrays E_sum and *path* for node 4 and the respective new values are stored in those.

After we are finished exploring all the parents of node 4, we go on to node 5. A similar procedure is followed for nodes 5, 6, 7, 8, 9 and the last node, 10. After completing the operation of the last node, we sort the indices stored in the I_sum array of node 10 in ascending order. Since all indices are unique, there will not be any tie while sorting. We arrange the entries in the arrays E_sum and *path* in the same order as their corresponding entries in the array I_sum have been sorted. The user then enters an energy threshold. The algorithm looks through the array E_sum sequentially and finds the first entry which has a value less than or equal to the energy threshold. The algorithm prints this value with the corresponding entries in the arrays I_sum and *path* for node 10. If no such energy sum value is found, then the algorithm prints that no such path exists.

Theorem 5.1. *Given a graph G of a linear work flow W, if a node i contains two paths, $path_1$ and $path_2$, having the same sum of indices till node i but the energy sum of $path_1 \leq$ the energy sum of $path_2$, then the entry for the $path_2$ can be discarded since it will never yield a smaller energy sum path than $path_1$.*

Proof. We will prove it by contradiction. Suppose we keep both the paths, $path_1$ and $path_2$ in the belief that at some later node, $path_2$ will go on to produce an energy sum which will be smaller than that produced by $path_1$ at the same node. Let there be a node j such that node i is the parent of node j, and the index value of node j be k. The index sum till node

j for $path_2$ will be the index sum of $path_2$ till node $i + k$. The sum of indices for $path_1$ till node j would be same as that for $path_2$ since we know that $path_1$ and $path_2$ have the same sum of indices till node i. So if j gets added to both the sum, the resultant value will always be the same. Now, let the energy sum along $path_1$ till node i be E_1 and that for $path_2$ till node i be E_2. Also, let the sum of energies consumed by the edge from node i to node j and by node j itself be E_j. The sum of energy consumed by $path_1$ when it adds node j to its path will be $E_1 + E_j$, and for $path_2$ will be $E_2 + E_j$. Since $E_1 \leq E_2$, $E_1 + E_j$ will also be $\leq E_2 + E_j$. Moreover, our aim is to give user the smallest energy consumed path for a particular sum of indices. So, it is of no use to keep an entry for $path_2$ after node i since for each and every succeeding node, it will produce the same sum of indices as that of $path_1$, but its energy sum will always be larger than that of $path_1$. □

Corollary 5.1. *Given a graph G of a linear work flow W, a path $path_1$ obtained at the last node corresponding to certain index sum is the path which consumes the least amount of energy for that particular index sum. In other words, we get the optimal path at the last node corresponding to their index sum.*

Proof. The proof is an extension of Theorem 5.1. At each node, if more than one path is found having the same sum of indices, then the one having the higher energy consumption is discarded. So at each node, we keep only those paths which have the least amount of energy consumed associated with them for a particular index sum. Therefore at the last node, we only get the paths corresponding to the least amount of energy consumed for a particular index sum value. □

Complexity of Algorithm 4

Let there be a total of T tasks that are represented in the linear work flow. Let each task have a maximum of K variants, including the cloud variant. The cloud variant and the first local variant have index value as 1 and the remaining variants have index value ranging from 2 to $K - 1$, where $K - 1$ is the last local variant. Assume i_1, i_2, i_3, ..., i_K are the K parents of node j. The maximum number of entries a node needs to store is $\{[(T-2)*(K-1)]+2\}*3$, since the maximum sum of indices at the last node of the work flow will be the sum of indices of the path which contains the variants of each task having index value equal to $K - 1$ (the last variant of each task). There will be $T - 2$ such tasks, and the remaining two tasks will be the native start task and the native end task which have only a single variant of value 1. Also, there is a multiplication by 3 since we need to store three arrays with respect to each node, *index_sum*, *energy_sum* and *path*. The space complexity is $[(T-2)*(K-1)]+2$.

Space complexity: $[(T-2)*(K-1)]+2$

The amount of time needed by a node will be the time needed to process $[(T-2)*(K-1)]+2$ paths from each of its parent nodes. In the worst case, there will be K parents. Also, this

precessing needs to be done for all nodes except for the start node, that is for $[(T-2)*K]+1$ nodes, with the assumption that all tasks have K variants but the last task has a single variant. So, the processing time will be added with the time to sort the entries in the last node in order of increasing indices sum which will take $\{[(T-2)*(K-1)]+2\}*log\{[(T-2)*(K-1)]+2\}$. The resultant time complexity is $\{\{[(T-2)*(K-1)]+2\}*\{[(T-2)*K]+1\}\}$ (for processing all the nodes) + $\{\{[(T-2)*(K-1)]+2\}*log\{[(T-2)*(K-1)+2]\}\}$ (for sorting the array *index_sum* at the last node), which can be approximated to $\{[(T-2)*(K-1)]+2\}*\{[(T-2)*K]+1\}$.

Time complexity: $\{[(T-2)*(K-1)]+2\}*\{[(T-2)*K]+1\}$

5.3 Algorithm for Concurrent Work flows

The algorithm to solve the problem scenario in the concurrent setting is given by Algorithm 5. The parameters used in Algorithm 5 are same as used for the linear algorithm.

Explanation of Algorithm 5

Line 1: The nodes need to be arranged in topological order.
Lines 7-8: In the original work flow, there might be edges from nodes immediately preceding the current node which might not be depicted in the topological ordering. So the energy of those edges will be added to the array *E_sum*.
The rest of the lines work similar to Algorithm 4.

Algorithm 5 starts by doing a topological sorting of all the nodes of the concurrent work flow. This is done to give a certain order of execution to all the nodes of the work flow. The ordering suggests one of the possible schedules of the work flow. A topological sort is done so that the dependencies among all the nodes are maintained. By converting the concurrent schedule to a linear topological schedule, there is no deviation from the actual problem, which we will prove through a theorem later in this section. In our example in Figure 5.3, the topological sorting will produce a linear schedule. One of the possible linear schedule is shown in Figure 5.4. Note that even though there are some edges missing and some new edges have popped up, the original set of edges between any two nodes are still present in the array *edge_e*. The remaining part of the algorithm is similar to that of the linear one with one difference. For each edge that was present in the original concurrent work flow but is absent in the topologically sorted work flow, the edge weight is added to a path when the destination node is encountered. For example, if in the original work flow, there was an edge from node i to node j but this edge is absent in the topologically ordered work flow, then the edge energy of edge ij is added during the processing of node j by looking up in the *edge_e* array. For the edges which have occurred due to the topological ordering but are actually absent in the concurrent work flow have the value 0 assigned to

Algorithm 5 Algorithm for solving Concurrent sequence of processes

do a topological ordering of all the nodes in the given workflow
while *all the parents of node_n have been explored* **do**
  for $i = 1$ *to entries*[*parent*] **do**
    $E\_sum \leftarrow E\_sum[parent][i] + edge\_e[parent][node\_n] + node\_e[node\_n]$
    $I\_sum \leftarrow I\_sum[parent][i] + index[node\_n]$
    Save the path in the *path* array
    for all the nodes i which have an edge to the current node in the original workflow **do**
      $E\_sum \leftarrow edge\_e[i][node\_n]$
    end for
    if I_sum already exists in an entry of array I_sum **then**
      keep the lower energy sum in the corresponding entry of the array E_sum and change the entry in the *path* array accordingly
    end if
  end for
  Explore the next parent
end while
Sort the array I_sum in ascending order and change the entries of E_sum and *path* accordingly
for $i \leftarrow 1$ *to entries*[*last_node*] **do**
  if *E_sum of that entry of the last_node node is* $\leq E\_threshold$ **then**
    print the entry's path, E_sum, I_sum
    EXIT
  end if
end for
print no path exists with energy less than the given $E\_threshold$

them in the *edge_e* array. So these edges are basically considered as edges without any energy consumption.

At the end of the algorithm, when we have completed the operation of the last node, we sort the indices stored in the *I_sum* array of the last node in ascending order. Also, we arrange the entries in the arrays *E_sum* and *path* in the same order as their corresponding entries in the array *I_sum* have been sorted. The user then enters an energy threshold. The algorithm looks through the array *E_sum* sequentially and finds the first entry which has a value less than or equal to the energy threshold. The algorithm prints this value with the corresponding entries in the arrays *I_sum* and *path* for the last node. If no such energy sum value is found, then the algorithm prints that no such path exists.

Theorem 5.2. *Given a graph G, a concurrent work flow C can be converted into a linear work flow L such that the schedule obtained from the linear work flow is a possible schedule that could have been chosen for the concurrent work flow.*

Proof. For the conversion, we use topological sorting. Because of this, the dependencies among the nodes are maintained. Since the topological sort arranges the nodes in one of the ways in which the control can reach from start to the end node in the concurrent graph, the linear schedule thus obtained is also a schedule of the concurrent graph. It is important to note that any one of the various schedules of the concurrent work flow will suffice to our requirement. □

Complexity of Algorithm 5

The space complexity of Algorithm 5 will be same as that for the linear algorithm. Let T be the total number of tasks and K be the maximum number of variants of a task, including the cloud variant. The first and the last tasks are native having a single local variant.

Space complexity: $O([(T-2)*(K-1)]+2)$

There will be some added time required for the initial topological ordering of the nodes. Also, while exploring each parent of a node, it needs to be looked up whether any edge exists so that its destination is the current node and it was present in the original concurrent work flow but is absent from the topologically sorted work flow.

The topological ordering can be done by a depth first search with an extra stack, which is linear in terms of the number of edges and the number of nodes. In worst case, it will be $T+[T*(T-1)]/2$ (since a completed graph will have $[T*(T-1)]/2$ edges). For looking up for a missing edge, in worst case $T-1$ nodes need to be searched and this time will be added to the processing time at each node. The processing time and sorting time (at the last node) will be same as that for the linear algorithm. The resultant time complexity is $\{T+[T*(T-1)]/2\}$ (for topological ordering) + $\{\{[(T-2)*(K-1)]+2\}*\{[(T-2)*K]+$

$1\} * (T-1)\}$ (for processing all the nodes) + $\{\{[(T-2)*(K-1)]+2\} * log\{[(T-2)*(K-1)+2]\}\}$ (for sorting the array *index_sum* at the last node), which can be approximated to $\{[(T-2)*(K-1)]+2\} * \{[(T-2)*K]+1\} * (T-1)$.

Time complexity: $\{[(T-2)*(K-1)]+2\} * \{[(T-2)*K]+1\} * (T-1)$

5.4 Experimental Evaluation

Cases	Minimum Energy path		Best QoS path		Energy consumed with best local variants *only*		Total number of unique paths
	Index Sum	*Energy Consumed (J)*	*Index Sum*	*Energy Consumed (J)*	*Index Sum*	*Energy Consumed (J)*	
5 Tasks, 2 Variants	8	6	5	13	5	13	4
10 Tasks, 2 Variants	12	13	10	19	10	35	9
15 Tasks, 2 Variants	17	14	15	20	15	60	13
20 Tasks, 2 Variants	21	9	10	20	10	85	19

Figure 5.6: Linear simulation by varying the number of tasks

Figure 5.6 shows a simulation obtained by varying the number of tasks for linear models. The number of variants of each task is kept fixed at 2, excluding the cloud variant. The column 'Energy consumed with best local variants only' signifies the energy consumed if no variants of the local task were used (not even the cloud variant). The 'Minimum energy path' column focuses on the path that consumed the least amount of energy in completing the tasks using the two local and one cloud variants. The 'Best QoS path' column shows the parameters of the path which provides the best quality of service path to complete the tasks, again using two local and one cloud variants. It can be seen that when we introduce the local and cloud variants, the amount of energy consumed gets decreased considerably (the 'Best QoS' column) for the same sum of indices or QoS parameter. This is due to the fact that there is a new cloud variant which has been introduced which provides an alternative path to the best variant path on the local device. The comparison of the 'Minimum energy' column with the 'Energy consumed with local variants only' column is shown in Figure 5.7.

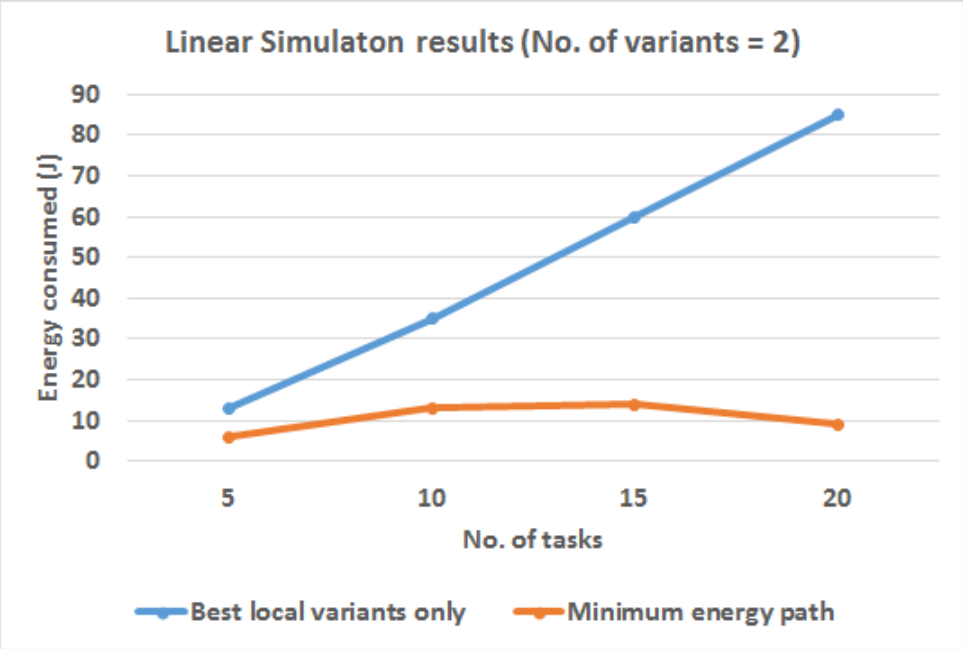

Figure 5.7: Linear simulation results (varying the number of tasks)

Cases	Minimum Energy path		Best QoS path		Energy consumed with best local variants *only*		Total number of unique paths
	Index Sum	*Energy Consumed (J)*	*Index Sum*	*Energy Consumed (J)*	*Index Sum*	*Energy Consumed (J)*	
5 Tasks, 2 Variants	8	6	5	13	5	13	4
5 Tasks, 3 Variants	11	5	5	13	5	13	7
5 Tasks, 4 Variants	12	3	5	13	5	13	10

Figure 5.8: Linear simulation by varying the number of variants

If the user has energy constraint, the the user can take a path which consumes the least amount of energy, which as seen from Figure 5.7 (lower curve) is much less than the path which only considers the best variants of the tasks on the mobile device (upper curve).

Next, we vary the number of tasks and the number of variants. The result for linear

Cases	Minimum Energy path		Best QoS path		Energy consumed with best local variants *only*		Total number of unique paths
	Index Sum	*Energy Consumed (J)*	*Index Sum*	*Energy Consumed (J)*	*Index Sum*	*Energy Consumed (J)*	
5 Tasks, 2 Variants	8	6	5	13	5	13	4
6 Tasks, 2 Variants	10	8	6	16	6	20	5
7 Tasks, 2 Variants	10	12	7	17	7	25	6
8 Tasks, 2 Variants	11	11	8	17	8	30	7
9 Tasks, 2 Variants	12	11	9	17	9	35	8

Figure 5.9: Concurrent simulation by varying the number of tasks

simulation is shown in Figure 5.8. The notable change is in the minimum amount of energy consumed to complete the process with multiple variants is significantly lower than if the best local variants were used. Also, with the increase in the number of variants, the number of unique paths also increases, which will enable the user to have more options to choose from.

Figure 5.9 and Figure 5.11 are the simulation of the concurrent models. The graph shown in Figure 5.10 shows the comparison between the least energy consuming path with all the variants (lower curve) against the path which considers the best local quality of service parameters for the tasks (upper curve). The outcome is similar to that of the linear one. This emphasizes the fact that use of variants can reduce the energy consumption to perform a set of tasks by decreasing the QoS parameter which will eventually aid a user to perform tasks in an energy constrained environment.

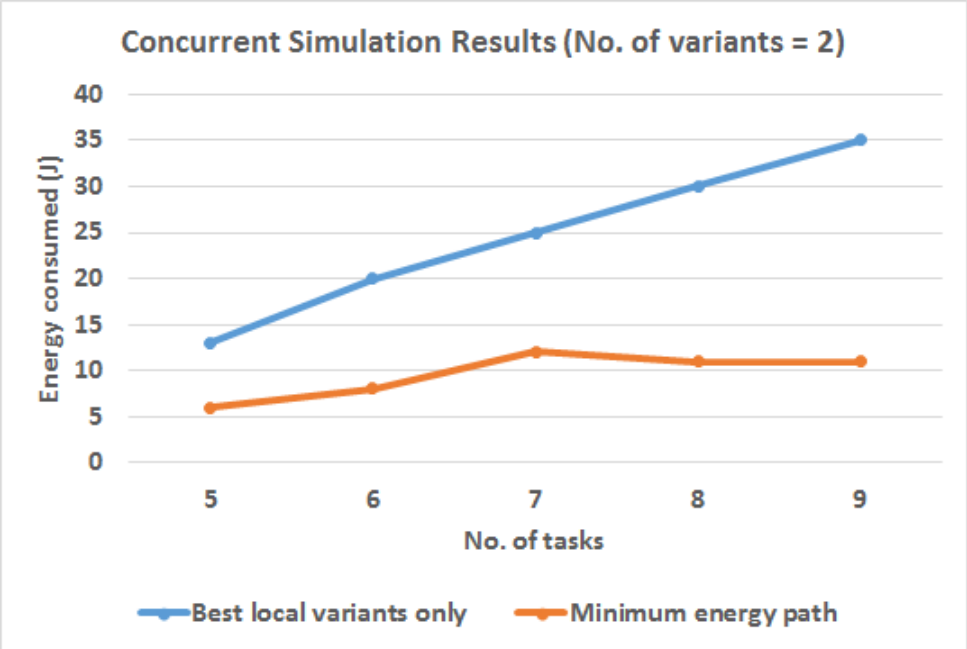

Figure 5.10: Concurrent simulation results (varying the number of tasks)

Cases	Minimum Energy path		Best QoS path		Energy consumed with best local variants *only*		Total number of unique paths
	Index Sum	*Energy Consumed (J)*	*Index Sum*	*Energy Consumed (J)*	*Index Sum*	*Energy Consumed (J)*	
5 Tasks, 2 Variants	8	6	5	13	5	13	4
5 Tasks, 3 Variants	8	4	5	13	5	13	7
5 Tasks, 4 Variants	10	3	5	13	5	13	10

Figure 5.11: Concurrent simulation by varying the number of variants

5.5 Conclusion

Variants of the original task give users the option to choose a lower energy path to complete their task by compromising on the Quality of Service parameter. This can be useful under

energy-constrained scenarios. We wish to extend our experimentation on image processing applications to see the impact of variants in most commonly used software tools.

Chapter 6

Conclusion and Future Work

In chapter 3, we proposed a scheduling algorithm which takes into account the percentage of CPU currently available for the use of an application. This gives a more realistic approach to energy analysis of an application and answers the question if it is possible to schedule a given application at a particular time. In chapter 4, we modified the scheduling algorithm presented in chapter 3 by creating clusters of applications. This is useful in the scenario where there are large number of applications and the scheduling will take a considerable amount of time to complete its operation. We also proposed an iterative refinement algorithm to *refine* the clusters. Finally in chapter 5, we suggested the use of variants of an application in a mobile device and a cloud variant. The scheduling algorithm can run on these variants and find several energy paths with different values of energy, from which the scheduler can choose the path which provides the best Quality of Service and satisfies a given energy constraint.

We are creating an android application which will combine the approaches presented in chapters 3, 4 and 5, and thus for any given application, answer the question if it is safe to schedule an application. Also for a set of applications having different variants, our android application will give the best path to schedule the given set of applications (in terms of Quality of Service), given an energy constraint.

Chapter 7

Disseminations out of this work

- A. Dash, and A. Banerjee, "When to schedule an application? An energy-aware decision," in *Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom '14)*, 2014.
- A. Dash, and A. Banerjee, "Application state refinement for scheduling applications on mobile devices," under review at *7th IEEE International Conference on Cloud Computing Technology and Science (CloudCom '15)*, 2015.

Bibliography

[1] Amobisense. Google Play app.

[2] Power tutor 2. Google Play app.

[3] Ermyas Abebe and Caspar Ryan. Adaptive application offloading using distributed abstract class graphs in mobile environments. *Journal of Systems and Software*, 85(12):2755–2769, 2012.

[4] Saeid Abolfazli, Zohreh Sanaei, Muhammad Shiraz, and Abdullah Gani. Momcc: market-oriented architecture for mobile cloud computing based on service oriented architecture. In *Communications in China Workshops (ICCC), 2012 1st IEEE International Conference on*, pages 8–13. IEEE, 2012.

[5] Pranav Balakrishnan and Chen-Khong Tham. Energy-efficient mapping and scheduling of task interaction graphs for code offloading in mobile cloud computing. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 34–41. IEEE Computer Society, 2013.

[6] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 280–293. ACM, 2009.

[7] A. Banerjee, H. S. Paul, A. Mukherjee, S. Dey, and P. Datta. A framework for speculative scheduling and device selection for task execution on a mobile cloud. In *ARMS-CC Workshop*, 2014.

[8] Marco Barbera, Sokol Kosta, Alessandro Mei, Vasile Perta, and Julinda Stefa. Mobile offloading in the wild: Findings and lessons learned through a real-life experiment with a new cloud-aware system. In *INFOCOM, 2014 Proceedings IEEE*, pages 2355–2363. IEEE, 2014.

[9] Kshitij Bhardwaj, Sreenidhy Sreepathy, Ada Gavrilovska, and Karsten Schwan. Ecc: Edge cloud composites. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*, pages 38–47. IEEE, 2014.

[10] Ruchita Bhargava et al. Energy consumption in data analysis for on-board and distributed applications. In *ICML*, 2003.

[11] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[12] Tracy D Braun, HJ Siegal, Noah Beck, Ladislau L Bölöni, Muthucumaru Maheswaran, Albert Reuther, James P Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen, et al. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pages 15–29. IEEE, 1999.

[13] Srdjan Čapkun, Jean-Pierre Hubaux, and Levente Buttyán. Mobility helps security in ad hoc networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 46–56. ACM, 2003.

[14] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, pages 21–21, 2010.

[15] Shuang Chen, Yanzhi Wang, and Massoud Pedram. A semi-markovian decision process based control method for offloading tasks from mobile devices to the cloud. In *Global Communications Conference (GLOBECOM), 2013 IEEE*, pages 2885–2890. IEEE, 2013.

[16] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[17] Byung-Gon Chun and Petros Maniatis. Dynamically partitioning applications between weak devices and clouds. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, page 7. ACM, 2010.

[18] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[19] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: Making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.

[20] Yong Cui, Shihan Xiao, Xin Wang, Minming Li, Hongyi Wang, and Zeqi Lai. Performance-aware energy optimization on mobile devices in cellular network. In *INFOCOM, 2014 Proceedings IEEE*, pages 1123–1131. IEEE, 2014.

[21] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G Bruce Berriman, John Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[22] Ding Ding, Siwei Luo, and Zhan Gao. A dual heuristic scheduling strategy based on task partition in grid environments. In *Computational Sciences and Optimization, 2009. CSO 2009. International Joint Conference on*, volume 1, pages 63–67. IEEE, 2009.

[23] Fangwei Ding et al. Monitoring energy consumption of smartphones. In *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, pages 610–613, Oct 2011.

[24] Mian Dong and Lin Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 335–348. ACM, 2011.

[25] Adam Dou, Vana Kalogeraki, Dimitrios Gunopulos, Taneli Mielikainen, and Ville H Tuulos. Misco: a mapreduce framework for mobile systems. In *Proceedings of the 3rd international conference on pervasive technologies related to assistive environments*, page 32. ACM, 2010.

[26] Heungsik Eom, Pierre St Juste, Renato Figueiredo, Omesh Tickoo, Ramesh Illikkal, and Ravishankar Iyer. Opencl-based remote offloading framework for trusted mobile cloud computing. In *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*, pages 240–248. IEEE, 2013.

[27] Adrienne Porter Felt, Serge Egelman, and David Wagner. I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, pages 33–44. ACM, 2012.

[28] Jason Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, WMCSA '99, 1999.

[29] Huber Flores, Pan Hui, Sasu Tarkoma, Yong Li, Satish Srirama, and Rajkumar Buyya. Mobile code offloading: from concept to practice and beyond. *Communications Magazine, IEEE*, 53(3):80–88, 2015.

[30] Wei Gao, Yong Li, Haoyang Lu, Ting Wang, and Cong Liu. On exploiting dynamic execution patterns for workload offloading in mobile cloud applications. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pages 1–12. IEEE, 2014.

[31] Apostolos Gerasoulis and Tao Yang. A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors. *Journal of Parallel and Distributed Computing*, 16(4):276–291, 1992.

[32] Shuai Hao et al. Estimating mobile application energy consumption using program analysis. ICSE '13, pages 92–101, 2013.

[33] Thomas A. Henzinger, Vasu Singh, Thomas Wies, and Damien Zufferey. Scheduling large jobs by abstraction refinement. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 329–342, New York, NY, USA, 2011. ACM.

[34] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. Mobile fog: A programming model for large-scale applications on the internet of things. In *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, pages 15–20. ACM, 2013.

[35] Edwin SH Hou, Nirwan Ansari, and Hong Ren. A genetic algorithm for multiprocessor scheduling. *Parallel and Distributed Systems, IEEE Transactions on*, 5(2):113–120, 1994.

[36] Yi-Hsuan Kao, Bhaskar Krishnamachari, Moo-Ryong Ra, and Fan Bai. Hermes: Latency optimal task assignment for resource-constrained mobile computing.

[37] A Khalifa and M Eltoweissy. Collaborative autonomic resource management system for mobile cloud computing. In *the Fourth International Conference on Cloud Computing, GRIDs, and Virtualization, Spain*, volume 20, pages 1–1, 2013.

[38] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953. IEEE, 2012.

[39] Karthik Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, (4):51–56, 2010.

[40] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4):406–471, 1999.

[41] Yu-Kwong Kwok, Anthony Maciejewski, Howard Jay Siegel, Arif Ghafoor, Ishfaq Ahmad, et al. Evaluation of a semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems. In *Parallel Architectures, Algorithms, and Networks, 1999.(I-SPAN'99) Proceedings. Fourth InternationalSymposium on*, pages 204–209. IEEE, 1999.

[42] Cheol-Hoon Lee and Kang G Shin. Optimal task assignment in homogeneous networks. *Parallel and Distributed Systems, IEEE Transactions on*, 8(2):119–129, 1997.

[43] David A Lifka. The anl/ibm sp scheduling system. In *Job Scheduling Strategies for Parallel Processing*, pages 295–303. Springer, 1995.

[44] Xue Lin, Yanzhi Wang, Qing Xie, and Massoud Pedram. Energy and performance-aware task scheduling in a mobile cloud computing environment. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 192–199. IEEE, 2014.

[45] Ehsan Ullah Munir, Jianzhong Li, Shengfei Shi, Zhaonian Zou, and Qaisar Rasool. A performance study of task scheduling heuristics in hc environment. In *Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 214–223. Springer, 2008.

[46] R. Murmuria et al. Mobile application and device power usage measurements. In *SERE 2012*, pages 147–156, 2012.

[47] Christos H Papadimitriou and Mihalis Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM journal on computing*, 19(2):322–328, 1990.

[48] Bumjoo Park and Namgi Kim. Secure and efficient communication method in rogue access point environments. *International Journal of Smart Home*, 7(4):37–46, 2013.

[49] Abhinav Pathak et al. Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof. EuroSys '12, pages 29–42, 2012.

[50] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. Odessa: enabling interactive perception applications on mobile devices. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 43–56. ACM, 2011.

[51] M Reza Rahimi, Nalini Venkatasubramanian, Sharad Mehrotra, and Athanasios V Vasilakos. Mapcloud: mobile applications on an elastic and scalable 2-tier cloud architecture. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, pages 83–90. IEEE Computer Society, 2012.

[52] Tariq Rashid. Clustering, 2007.

[53] Sonal Saha and Binoy Ravindran. An experimental evaluation of real-time dvfs scheduling algorithms. In *Proceedings of the 5th Annual International Systems and Storage Conference*, page 7. ACM, 2012.

[54] Zohreh Sanaei, Saeid Abolfazli, Abdullah Gani, and Rajkumar Buyya. Heterogeneity in mobile cloud computing: taxonomy and open challenges. *Communications Surveys & Tutorials, IEEE*, 16(1):369–392, 2014.

[55] Mahadev Satyanarayanan, Grace Lewis, Edwin Morris, Soumya Simanta, Jeff Boleng, and Kiryong Ha. The role of cloudlets in hostile environments. *Pervasive Computing, IEEE*, 12(4):40–49, 2013.

[56] Aaron Schulman, Vishnu Navda, Ramachandran Ramjee, Neil Spring, Pralhad Deshpande, Calvin Grunewald, Kamal Jain, and Venkata N Padmanabhan. Bartendr: a practical approach to energy-aware cellular data scheduling. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, pages 85–96. ACM, 2010.

[57] Mohsen Sharifi, Somayeh Kafaie, and Omid Kashefi. A survey and taxonomy of cyber foraging of mobile devices. *Communications Surveys & Tutorials, IEEE*, 14(4):1232–1243, 2012.

[58] Cong Shi, Vasileios Lakafosis, Mostafa H Ammar, and Ellen W Zegura. Serendipity: enabling remote computing among intermittently connected mobile devices. In *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*, pages 145–154. ACM, 2012.

[59] Muhammad Shiraz, Abdullah Gani, Rashid Hafeez Khokhar, and Rajkumar Buyya. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. *Communications Surveys & Tutorials, IEEE*, 15(3):1294–1313, 2013.

[60] Harold S Stone. Multiprocessor scheduling with the aid of network flow algorithms. *Software Engineering, IEEE Transactions on*, (1):85–93, 1977.

[61] Chris Thompson, Hamilton Turner, and Jules White. Analyzing mobile application software power consumption via model-driven engineering.

[62] Tim Verbelen, Tim Stevens, Filip De Turck, and Bart Dhoedt. Graph partitioning algorithms for optimizing software deployment in mobile cloud computing. *Future Generation Computer Systems*, 29(2):451–459, 2013.

[63] Yanzhi Wang, Xue Lin, and Massoud Pedram. A nested two stage game-based optimization framework in mobile cloud computing system. In *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*, pages 494–502. IEEE, 2013.

[64] Liyao Xiang, Shiwen Ye, Yuan Feng, Baochun Li, and Bo Li. Ready, set, go: Coalesced offloading from mobile devices to the cloud. In *INFOCOM, 2014 Proceedings IEEE*, pages 2373–2381. IEEE, 2014.

[65] Jia Yu, Rajkumar Buyya, and Chen Khong Tham. Cost-based scheduling of scientific workflow applications on utility grids. In *e-Science and Grid Computing, 2005. First International Conference on*, pages 8–pp. IEEE, 2005.

[66] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language. In *OSDI*, volume 8, pages 1–14, 2008.

[67] Osmar R Zaïane. Principles of knowledge discovery in databases-chapter 8: Data clustering. In *& Shantanu Godbole data mining Data mining Workshop 9th November*, 2003.

[68] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 105–114. ACM, 2010.

[69] Yunkai Zhang, Renyu Yang, Tianyu Wo, Chunming Hu, Junbin Kang, and Lei Cui. Cloudap: Improving the qos of mobile applications with efficient vm migration. In *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*, pages 1374–1381. IEEE, 2013.