5-1-2014

An Energy-Efficient, Time-Constrained Scheduling Scheme in local mobile cloud

Ting Shi
University of Nevada, Las Vegas

AN ENERGY-EFFICIENT, TIME-CONSTRAINED SCHEDULING SCHEME IN LOCAL MOBILE CLOUD

by

Ting Shi

Bachelor of Engineering in Electrical Engineering

Tianjin University

2008

A thesis submitted in partial fulfillment

of the requirements for the

Master of Science in Engineering – Electrical Engineering

Department of Electrical and Computer Engineering

Howard R. Hughes College of Engineering

The Graduate College

University of Nevada, Las Vegas

January 2014

UNLV
UNIVERSITY OF NEVADA LAS VEGAS

THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Ting Shi

entitled

An Energy-Efficient, Time-Constrained Scheduling Scheme in Local Mobile Cloud

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering
Department of Electrical and Computer Engineering

Mei Yang, Ph.D., Committee Chair

Yingtao Jiang,Ph.D.,Committee Member

EbrahimSaberinia, Ph.D., Committee Member

Hui Zhao, Ph.D., Graduate College Representative

Kathryn HausbeckKorgan, Ph.D., Interim Dean of the Graduate College

May 2014

ABSTRACT

Mobile devices have limited resource, such as computation performance and battery life. Mobile cloud computing is gaining popularity as a solution to overcome these resource limitations by sending heavy computation to resourceful servers and receiving the results from these servers. Local mobile clouds comprised of nearby mobile devices are proposed as a better solution to support real-time applications. Since network bandwidth and computational resource is shared among all the mobile devices, a scheduling scheme is needed to ensure that multiple mobile devices can efficiently offload tasks to local mobile clouds, satisfying the tasks' time constraint while keeping low-energy consumption. Two critical challenges need to be solved: (1) estimation of the energy consumption and completion time for tasks to be scheduled, (2) schedule the tasks from multiple source nodes to an appropriate device to accomplish the computation and receive the results.

In this thesis, the adaptive probabilistic task scheduler for local mobile clouds is proposed. The scheduler relies on periodic network messages to discover neighboring computation and network resources. It first estimates the completion time and energy consumption at each potential processing node. Next, it schedules the current task to the proper processing node in a probabilistic way and adaptively adjusts its time margin to improve performance under the unpredictable network condition. Comparing with other existing scheduling schemes, the experimental results confirm that the proposed scheduler achieves highest task completion rate and the lowest average energy per successful task. In addition, the proposed scheduler is able to accommodate different types of tasks and network scenarios.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Mobile devices have become a crucial part of our daily life nowadays. More than 1.99 billion mobile phones and tablets will be sold worldwide in 2013, according to Gartner's analysts [1]. As the capabilities of mobile devices advance (in terms of CPU power, network connectivity and sensors), people increasingly use them for the tasks such as emailing, web surfing, gaming etc. Although there have been many advances in technology, mobile devices will still be resource poor, as restrictions on weight, size, battery life, and heat dissipation impose limitations on computational resources and make mobile devices more resource constrained than their non-mobile counterparts. One solution to overcome these resource limitations is mobile cloud computing, which allows the mobile device to offload the tasks to more powerful resource devices (i.e., servers), as shown in Figure 1.

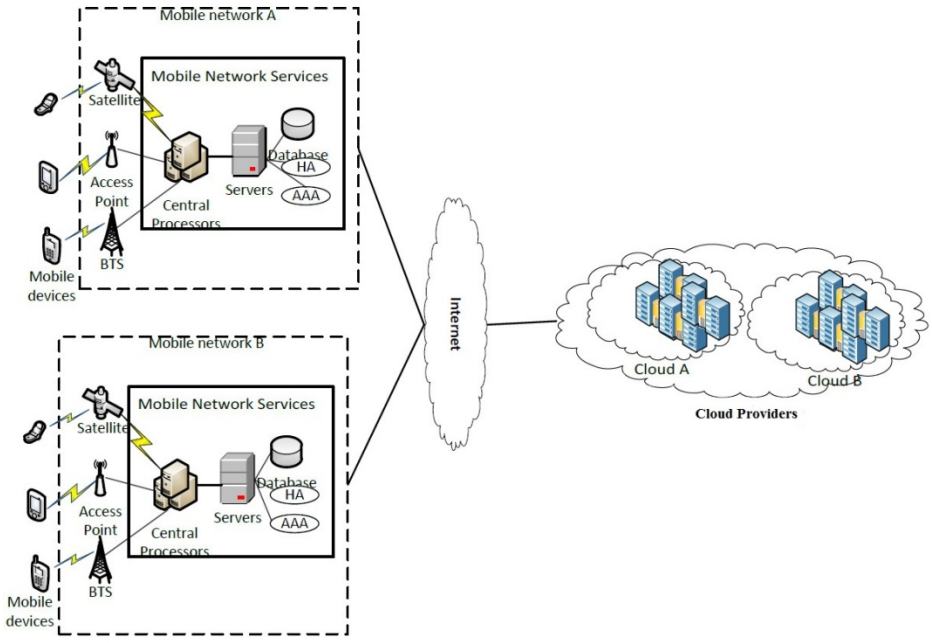


Figure 1, Mobile cloud computing architecture

A. Overview of mobile cloud computing

Offloading aims at augmenting the mobile device's capabilities by using resource providers other than the mobile device itself to host the execution of mobile applications. Since offloading migrates computation to more resourceful computers, it involves making decisions regarding whether and what computation to offload. Therefore, the key challenge is to carry out a cost-benefit analysis to weigh the benefits of offloading against the cost of remote execution with user specific requirement. The decisions are usually made by analyzing parameters including bandwidths, server speeds, available memory, server loads and the amount of data exchanged between servers and mobile systems.

To implement the offloading scheme, two components are used in the mobile device as introduced in [2]. One is profiler, the other one is scheduler. The profiler measures the device characteristics at initialization time, and it continuously monitors the device and network characteristics which can often change. A stale measurement may cause a wrong offloading decision. The scheduler uses the data collected by the profiler as input to a task scheduling problem to determine whether to offload the current task and which resource provider to choose.

Offloading becomes an attractive solution for meeting stringent response time requirement in mobile systems as applications become increasingly complex. For example, in real-time moving object recognition and tracking system [3], a robot needs to recognize an object then adjust its speed and direction to track the object's movement. If the robot's processor is too slow, the recognition computation might not be able to be completed before the object moves out of the surveillance range. Much research has been done in mobile cloud to enable the mobile devices to run real-time applications such as

object recognition [4], augmented reality [5], disaster forecast [6], and real-time video coding [7].

Energy consumption is another factor which needs to be taken into consideration when designing mobile cloud systems. Even though battery technology has been steadily improving, it has not been able to keep up with the rapid growth of power consumption of these mobile systems. Energy is still a primary constraint for mobile systems. Therefore, offloading is also used to save energy by migrating the computation tasks with high energy consumption to servers. Many scheduling algorithms have been proposed to make offloading decisions to improve performance or save energy [8] [9].

B. Motivation

One type of mobile cloud architecture is to connect the mobile device to the remote server, which is typically far away from the mobile user. A drawback of this architecture is that the high Wide Area Network (WAN) latency makes it unsuitable for real-time applications. Another type of mobile cloud architecture is to consider other mobile devices themselves also as resource providers of the cloud, forming a mobile peer-to-peer network. This network is built based on the Mobile Ad Hoc Network (MANET). An MANET does not rely on pre existing infrastructure, such as routers in wired network. Instead, each mobile node participates in routing by forwarding data for other nodes. To reduce the latency in offloading process in MANETs, it is preferred to offload tasks to nearby devices. Therefore the second type of mobile cloud architecture will be more suitable to support real-time applications, which demand faster responsiveness.

In this thesis, we refer to the second type of mobile cloud architecture as the 'local mobile cloud' as shown in Figure 2. Task scheduling schemes have been investigated in [5][6][10] to make offloading more beneficial in local mobile cloud. But existing scheduling schemes have several crucial constraints and limitations, including: (a) serving only one source node, instead of the entire network [5][11]; and (b) assuming that the detailed processing information of each participating mobile node, such as the status of local task queue and real-time resource utilization, is known to all the other nodes [12]. Furthermore, such a scheduler is also supposed to work in a decentralized manner and be able to dynamically adapt to the changes in the network through time.

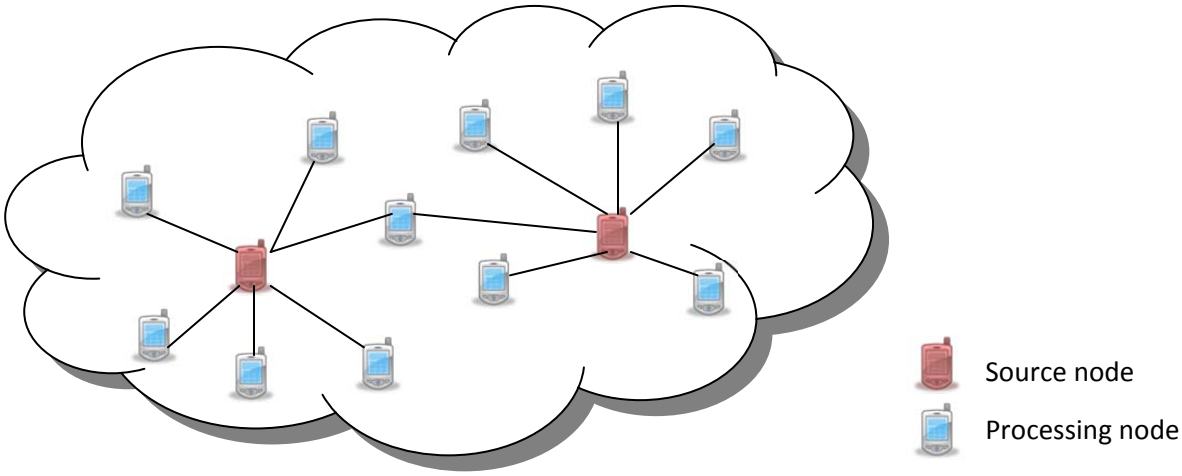


Figure 2, Local mobile cloud architecture

The focus of this thesis is to propose a distributed task scheduling scheme which allows the source nodes to offload tasks to nearby devices in a local mobile cloud, satisfying the tasks' time constraint while keeping low-energy consumption for real-time applications with soft time constraint. To develop such a scheduling scheme, two critical

challenges need to be solved: (1) estimation of the energy consumption and completion time for tasks to be scheduled according to the current network condition, (2) schedule and assign the tasks from multiple users to appropriate devices to accomplish the computation.

C. Introduction to this thesis

In this thesis, the task scheduling problem under time constraint is investigated. A QoS OLSR-based scheme is used to disseminate the workload, computation ability and energy status of each potential target node periodically so that context change is updated on time. An adaptive probabilistic task scheduler is proposed, which is a promising approach for real-time applications due to its scalability and flexibility in local mobile cloud. The scheduler relies on periodic network messages to discover neighboring computation and network resources and keep track of topological changes as well as resource updates in the network. It first estimates the completion time and energy consumption at each potential processing node. Next, it schedules the current task to the proper processing node in a probabilistic way and adaptively adjusts its time margin to improve performance under the unpredictable network condition. A simulation model of local mobile clouds has been implemented on OMNET++ [13] to comprehensively evaluate the performance of the proposed scheduler and analyze the overhead cost and the improvement gained.

This thesis is organized as follows. Chapter 2 summarizes the related work done for mobile clouds and also lists challenges to provide cloud services in MANET.

Chapter 3 first introduces the network model and task model. Then the task scheduling problem in local mobile clouds is formulized. A motivating example is given to explain the need for the efficient scheduler in local mobile clouds.

In Chapter 4, the adaptive probabilistic scheduling scheme in local mobile clouds is proposed. It has two parts, 1) the resource discovery scheme, and 2) the adaptive probabilistic scheduling algorithm. Two parts are described separately. An illustrative example is given to demonstrate the difference between the proposed scheme and other scheduling algorithms.

In Chapter 5, the simulation environment and parameter settings in our simulation is introduced. The performance results are presented to compare the proposed task scheduling scheme with other task scheduling schemes. Experiment 1 evaluates the overhead generated by the resource discovery and how the resource discovery scheme affects the performance of different schedulers in the local mobile cloud. Experiment 2 shows how the number of the mobile nodes in the local mobile cloud affects the performance. Experiment 3 compares the performance of different schedulers in stationary scenario and in mobile scenario. Experiment 4 shows the performance of different schedulers with different types of the tasks.

Chapter 6 concludes the thesis and suggests the future work.

CHAPTER 2 LITERATURE REVIEW

In this chapter, the existing research on mobile cloud computing will be reviewed, including the following topics: mobile cloud architectures, task-scheduling in mobile cloud system and context awareness in mobile clouds.

A. Mobile cloud architectures

There are two typical types of mobile cloud architectures nowadays. The first one is to connect the mobile device to the remote cloud as in Figure 1. The remote cloud is defined as a powerful server or a cluster of computer hardware and software that offer the services to the mobile device users. By leveraging infrastructures such as Amazon's EC2 cloud [14] or Apple iCloud [15], computationally expensive tasks can be offloaded to the cloud so that the capabilities of mobile devices get improved. For example, Apple Siri [16] runs its sophisticated voice recognition feature remotely and then returns the result to the user. There is large amount of research regarding implementing this architecture [2] [17] [18]. However, as mentioned in Chapter 1, this architecture suffers from the long latency. A detailed analysis on why long WAN latencies are a fundamental obstacle in mobile clouds can be found in [19]. Another drawback of this architecture is that it relies on the Internet access. 3G and WLAN are two most popular ways that mobile devices connect to the internet. 3G covers larger area but consumes more energy while WLAN consumes less energy but the transmission range is limited [20]. It is not practical to expect that a good Internet connection is always available. Let alone in battle field, disaster scene and rural area where no Internet access is present but intensive computations are still needed.

Therefore, a local mobile cloud would be a better alternative to the remote cloud. A local mobile cloud is defined as the collective resources of the various mobile devices in the local vicinity which are utilized as service providers. The advantages of offloading work with local nearby resources versus a remote cloud would be:

- It does not rely on the Internet to connect to remote servers;
- It can be easily deployed in different scenario;
- Connection to nearby devices results in less latency.

Hyrax [10] explores the possibility of using a cluster of mobile phones as resource providers and shows the feasibility of such a local mobile cloud. It is implemented for the Android smartphones based on ported Hadoop [21]. In Hyrax, a central server is connected to each mobile device and coordinates data and tasks without doing any of the processing. The mobile devices communicate with each other on an isolated 802.11g network. Another framework based on Hadoop is presented in [22]. The system's offloading manager module organizes sending and receiving tasks to and from processing nodes and creating virtual machines on them. The processing time is actually less than if executed on a single mobile device. Energy consumption is claimed to be reduced since energy consumption is proportional to the processing time.

A cloudlet architecture is presented in [5] that not only provides a fixed infrastructure with the WiFi access point, but also enables ad hoc discovery of devices in the vicinity to share resources among each other. The cloudlet infrastructure is not fixed, and devices can join and leave the cloudlet at runtime. A mobile device connects to the network can register as a service provider. The proposed cloudlet architecture provides a middleware framework to support real-time applications, which are managed at

component level. Each component is a part of codes need to be executed. When a performance constraint violation is detected, actions can be taken such as calculating a new deployment (i.e. offloading some resource-intensive components) or adapting component configurations (i.e. lowering component quality). The result shows that compared with connecting to a remote cloud, cloudlet is more suitable for the real-time applications.

In above work, there are two common methods related to offloading or/and sharing work from mobile devices, partitioning applications and Virtual Machine (VM) migration, which are introduced in [2][9][17], respectively. Partitioning applications into tasks or components increase the manageability and scheduling efficiency. VM migration refers to transferring the memory image of a VM from a source device to the destination server without stopping its execution [23]. These are the enabling techniques for offloading in mobile clouds. However, they are not the focus of this thesis. Our study is focused on the task scheduling under the premise that the application could be partitioned into independent 'offloadable' tasks.

B. Task-scheduling in mobile cloud system

a. Scheduler architecture overview

The architecture of a scheduler plays an important role in determining the scalability, autonomy, and performance of the system. Majorly there are two categories: centralized and decentralized.

In a centralized scheduling architecture, such as in [10], the scheduling decisions are made by a central controller based on the complete and reliable knowledge of all the participating nodes. The centralized scheduler suffers from the following problems:

- Single point failure problem;
- link to the central node is shared among all participating nodes which causes the bottleneck problem;
- unrealistic deployment of the central node in mobile scenarios. Only the mobile nodes within the transmission range of the central node can join the local mobile cloud, which limits the mobility of the network.

Thus, centralized scheduler is not adequate for the mobile cloud system because of the nature of the dynamic environment in local mobile clouds.

In contrast, decentralized schedulers negate the limitations of centralized schedulers with respect to scalability, autonomy, and most importantly the adequacy for the dynamic local mobile cloud. It does not rely on one single central node. The decentralized scheduler can work as long as a group of mobile nodes are connected, which make the deployment much easier. A decentralized scheduling approach assumes that each participating node is autonomous and has its own controller that derives its scheduling decision based on its policies. However, if the decisions are taken by several independent nodes, it might be the case that these units aim at optimizing their own objectives rather than the performance of the system as a whole. Such situations call for models and techniques that take the strategic behavior of individual units into account, and simultaneously keep an eye on the global performance of the system.

b. Scheduling objective in mobile cloud system

Generally, schedulers generate the mapping of tasks to resources based on some particular objectives. In a local mobile cloud, the assignment of computational tasks to different devices plays a vital role in energy conservation and performance

improvement. It is the task scheduler's responsibility to make sure the mobile device could benefit from offloading. Various cost/benefit studies have different criteria on whether to offload computations to the server. The commonly used scheduling objectives in a mobile cloud computing environment are related to the tasks' completion time and energy consumption.

The Eqn. (1) has been used in different paper [3][24] as a condition for offloading to improve tasks completion time:

$$\frac{w}{s_m} > \frac{d_i}{B} + \frac{w}{s_s} \tag{1}$$

Suppose *w* is the amount of computation and d_i is the size of data needs to be transmitted for remote execution. Let s_m be the speed of the mobile device and s_s be the speed of the server. *B* is bandwidth between the mobile device and the server. The left side is the time needed to execute the task on the mobile device itself. The right side is the time needed to execute the task on the server plus the transmission time. Offloading can improve performance when execution can be completed faster at the server.

An analysis aiming to save energy is provided in [25]. Similar to Eqn. (1), suppose the task's computation requires *C* instructions. Let *S* and *M* be the execution speed in instructions per second, of the cloud server and the mobile device, respectively. The server and mobile device exchange *D* bytes of data and the network bandwidth is *B* bps. The mobile device consumes, in watts, P_c for computing, P_i while being idle, and P_{tr} for sending and receiving data. When the server performs the computation, the amount of energy saved is given by

$$P_c \times \frac{C}{M} - P_i \times \frac{C}{S} - P_{tr} \times \frac{D}{B} \tag{2}$$

Different scheduling schemes with more complex criteria are proposed in several other papers. A customizable task scheduler is proposed in [11] using MapReduce framework for local mobile clouds. It is customizable because the user can optimize multiple objectives such as power consumption and (or) throughput through adjusting the corresponding coefficients in the objective function in Eqn. (3), where $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$ are variable coefficients, F_1 is a function of the task processing time on node j, F_2 is a function of task queuing time on node j, F_3 is a function of the communication cost of sending a task from node i to node j, F_4 is a function of battery level on node j, F_5 is the energy consumption of sending a task from node i to node j.

$$F(i) = Min[\alpha_1 \times F_1(PT_j) + \alpha_2 \times F_2(Q_j) + \alpha_3 \times F_3(Comm[ij]) + \alpha_4 \times F_4(E_j, C_j) + \alpha_5 \times F_5(EComm[ij])] \quad for\ i, j = 1,2 \dots n, j \neq i \tag{3}$$

Similar work has been done in [12]. The objective function has two parts: delay and energy consumption. Delay includes the task execution time and queuing time at the processing node and communication time. Energy consumption consists of computation energy consumed at the processing node and communication energy along the path. The scheduler can be delay constrained, energy constrained or delay/energy constrained.

c. Scheduling algorithm

However, the aforementioned work is only concerned with one mobile device connecting to one server scenario. In a local mobile cloud system, there could be multiple source devices and multiple servers.

In [26], a decentralized dynamic scheduling approach entitled the community aware scheduling algorithm (CASA) is introduced for mobile grids. It includes the task submission phase and the dynamic scheduling phase, which work together to ensure both

a rapid task distribution and an optimized rescheduling process. In the task submission phase, the scheduler sends the 'offloading' request containing the task information to nearby nodes. Nearby nodes which are able to execute this task will reply an 'accept' message to the source node, in which the response time of the task is included. In this paper, the task response time represents the time used between a task's arrival time until the time when the execution result is received. Considering multiple requests from different source nodes could exist simultaneously, if all nodes greedily select the processing node offering the shortest task response time for the task assignment, then that processing node could receive an imbalanced amount of tasks within one CASA scheduling cycle, which might increase the response time for some received tasks. In order to avoid this effect, the CASA scheduler adopts a probabilistic approach, wherein nodes with better computing power are prone to have a chance of receiving more tasks, but nodes with less power still have the probability of being selected as the processing node. The rescheduling phase is to keep the previous scheduling decision optimized by allowing queued tasks to be rescheduled in accordance with the unexpected network delay, resource overhead, and task status modification. Comparing with the centralized scheduling scheme [27] and the greedy based scheduling algorithm [28], the use of CASA can lead to a 30% – 61% better average task slowdown, and a 68% – 86% shorter average task waiting time in a decentralized scheduling manner without requiring detailed real-time processing information from participating nodes.

This work gives us the insight of how to handle multiple source devices. One thing not clear is that how each node can get the detailed information from the participating nodes to maximize the offloading benefit or to successfully offload.

C. Context awareness in mobile cloud

In this thesis, context means resource information of nearby nodes, such as power information, bandwidth and computational ability. The scheduler with context-awareness is able to use contextual information to make scheduling decision and automatically reconfigure their configurations to adapt to the context. Context awareness is more difficult to achieve in wireless ad hoc networks than in their wired counterparts, because the wireless bandwidth is shared among adjacent nodes and the network condition changes dynamically.

Within the field of local mobile clouds, there are, broadly, two different approaches to discover the context information. Reactive schemes request other nodes' information when needed. A node trying to transmit a packet may have to wait for the completion of resource discovery. The mobile cloud systems in which source nodes send requests to processing nodes and wait for responses work in the reactive way, such as [12][26]. As mentioned in [29], a reactive scheme has small resource discovery overheads but a longer react time. On the other hand, proactive schemes determine the resource status of various nodes in the network in advance, so that the context information is already present whenever needed. Resource discovery overhead is large in such schemes as one has to send control messages to update all context information periodically. Scheduling decision making is faster as the context information is already present.

There is a lot of research in reactive resource discovery scheme in Ad hoc network as introduced in [30]. However, considering the tight time constraint set by the application, a proactive way is very attractive because it is always ready to use whenever

needed. In [6], A QoS OLSR [31][32] based routing scheme is proposed to map computationally intensive real-time application to all nearby available resources which are heterogeneous and with limited computational capabilities. This work shows the feasibility of providing cloud services in MANET. The authors modified the original QoS OLSR by adding resource information including CPU type, CPU utilization percentage, the allocated memory and battery levels to control messages. Control message are broadcasted to the whole network periodically so that all nodes are aware of the resource information of other nodes. However, how frequent the routing messages are sent is not given in this paper. The context information is updated upon receiving the routing messages. The more frequent the routing messages are, the more accurate the context information is. A frequent context information update could be big overhead and the routing packets' number and size have a major impact on the local mobile cloud's performance. However, no detailed evaluation about the overhead is given in this work.

To summarize, current research and implementation work have two crucial constraints and limitations, including: (a) scheduling for serving one source node, instead of the entire network; and (b) assuming the detailed resource information of each participating node, such as the status of local task queue and real-time resource utilization, is known. These impede the implementation of mobile cloud system.

CHAPTER 3 PROBLEM STATEMENT

A. Notations and assumptions

Table 1 lists the notations used in this thesis.

Table 1, Notations

Symbol	Description
V	Set of all wireless nodes in the local mobile cloud
E	Set of all wireless links in the local mobile cloud
$G(V, E)$	Undirected topology graph that is composed of node set V and edge set E
$mips_u$	Average processing speed of node u in terms of millions of instructions per second (mips)
e_u	Average energy consumption per million instructions of node u
r_t	Radio transmission range of node u
$B_{u,v}$	Bandwidth (in bps) between node u and node v
e_t	Average energy consumption of node u to transmit one byte
e_r	Average energy consumption of node u to receive one byte
$t_{q,u}$	Queuing time of node u
J	Set of tasks arriving at V
D_j	Data size of task j
C_j	Computation amount of task j in number of instructions
T_j	Time constraint set for task j
t_{margin}	Time margin used when comparing the estimated task completion time with T_j
P	Set of all processing nodes in the local mobile cloud, $P \subseteq V$
S	Set of all source nodes in the local mobile cloud, $S \subseteq V$

Assumptions:

1. The tasks are assumed to be computationally intensive, mutually independent, and can be executed at any participating processing mobile devices. As soon as a task arrives,

it must be assigned to one mobile device for processing. After a task is executed at the processing node, the result will be returned to the originating device.

2. The size of data D_j needs to be transferred from the source node to the destination node is the same as the size of result returned from the destination.

3. The IEEE 802.11g communication protocol is adopted. The transmission power e_t, the receiving power e_r and the transmission range r_t are assumed to be the same among all the nodes. The maximum bandwidth B_{max} is 11Mbps and it is shared among adjacent nodes.

4. All nodes in the local mobile cloud are randomly distributed. Any two nodes are connected either directly or in an ad hoc way.

B. Network and task models

- Network models

Definition 1.A MANET that consists of a number of wireless nodes can be modeled by an undirected communication graph *G(V, E).* Given a node $u \in V$ and a node $v \in V$, we have $(u, v) \in E$, if and only if $dis(u, v) \leq r_t$, where $dis(u, v)$ is the Euclidean distance between node u and node v. That is, to establish a direct communication between any two nodes, the distance between them has to be within their radio range r_t.

The computational ability of the processing node u is denoted by $mips_u$ (Million instructions per second). According to [33], Eqn. (4) gives that the power e consumed by a CMOS processor, in watts, is equal to the activity factor α of the system (percentage of gates that switch for each cycle, on average 50%) multiplied by the capacitance C of the CPU, the voltage V squared, the frequency f.

$$e = \alpha C V^2 f \quad (4)$$

However, historical data [34] suggests that power on modern processors is proportional to the square of the duty cycle. Therefore, for this thesis we will use the square relation in Eqn. (5), assuming that the CPU's power consumption per instruction e_u is proportional to the square of its speed, $mips_u$.

$$e_u \propto {mips_u}^2 \quad (5)$$

In each node, $t_{q,u}$ is the queuing delay, which refers to the waiting time when the task is placed at the end of the queue until the moment the task is processed.

- Task models

J is a set of tasks arriving at different source nodes, for a task $j \in J$, it has the following attributes:

1) D_j, size of data to be transferred between the source node and the processing node;

2) C_j, amount of computations to be processed;

3) T_j, time constraint of task j.

All these attributes of task j are known to the source node when j arrives.

C. Problem statement

With the network model and task model, the energy consumption and completion time can be calculated. The energy consumption of executing a task includes two parts:

1) Computation energy, which is the energy dissipated for executing the task by the processing node u;

$$Computation\ Energy_j = e_u \times C_j \quad (6)$$

2) Communication energy, which is the energy consumed in communication for the offloading process. The communication energy dissipated is proportional to the amount

of data transmitted or received. $H(j)$ is the hop count from the source node to the processing node.

$$Communication\ Energy_j = 2 \times H(j) \times (e_t + e_r) \times D_j \quad (7)$$

The total energy $Task\ Energy_j$ consumed by task j is the summation of these two types of energy:

$$Task\ Energy_j = Computation\ Energy_j + Communication\ Energy_j \quad (8)$$

The total energy consumed by all the n tasks is:

$$Total\ Energy = \sum_{j=1}^{n} Task\ Energy_j \quad (9)$$

The completion time of a task contains three parts:

- $t_{q,u}$: queuing time at the processing node u, which is the waiting time before task can be executed;
- $t_{execution}$: the execution time for the task at the processing node u;

$$t_{execution} = C_j / mips_u \quad (10)$$

- $t_{transmition}$: the time needed to transmit the data between the source and destination nodes.

$$t_{transmition} = 2 \times (H(j) - 1) \times D_j / B_{u,v} \quad (11)$$

The completion time of the task j when it is offloaded to node u is the total of the three parts:

$$Task\ completion\ time_j = t_{q,u} + t_{execution} + t_{transmition} \quad (12)$$

A task is successfully scheduled if it is completed before its deadline, satisfying Eqn. (13); otherwise, it is failed.

$$Task\ completion\ time_j < T_j \tag{13}$$

Let n be the total number of tasks. We define the following two metrics to evaluate the task scheduling result.

$$Completion\ Rate = Number\ of\ Successful\ Tasks/n \tag{14}$$

$$Average\ Energy\ per\ \ Successful\ Task = \frac{Total\ Energy}{Number\ of\ Successful\ Tasks} \tag{15}$$

We formulize the scheduling problem in a local mobile cloud as: Given the mobile network *G(V, E)* with a set of source nodes *S* a a set of tasks *J* to be scheduled, then schedule these tasks from one or multiple source nodes to processing nodes, with the objective of

Minimize: $Average\ Energy\ per\ Successful\ Task$

Subject to: $B_{u,v} \leq B_{max}, \forall e = (u, v) \in E,$ (16)

$$\sum\nolimits_{u \in P} x_{j,u} = 1, \forall j \in J, x_{j,u} \in (0,1) \tag{17}$$

$$P \subseteq V, S \subseteq V, P \cap S = \varnothing, P \cup S = V \tag{18}$$

Constraint (16) ensures no link capacity is violated. Constraint (17) ensures that a task is scheduled only once. Constraint (18) states that a node in the local mobile cloud is either a processing node or a source node.

D. Motivation example in local mobile clouds

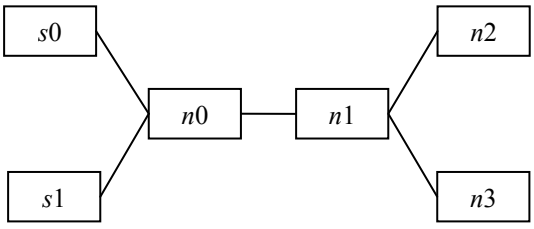


Figure 3, Sample local mobile cloud

We use one simple example to illustrate the motivation for an effective scheduler in the local mobile cloud. Consider the case in Figure 3, there are two source nodes running real-time applications. The applications can be partitioned into tasks, which are computationally intensive and are executable in all processing nodes *n*0 to *n*3. The challenges are listed as follows.

- Scheduling of tasks from multiple source nodes schedule to the same processing node

Having the same context information about the network, it is possible that multiple source nodes schedule tasks to the same processing node when they make scheduling decision at the same time. Assuming all tasks need to be executed in processing node serially in a first come first serve way, this will cause the unpredictable completion time for the tasks which arrive later. In Figure 3, in terms of task completion time, $t(n_3) < t(n_2) < t(n_1) < t(n_0)$. If both *s*0 and *s*1 schedule their current task to *n*3, it is possible that *n*3 is only able to complete one task before deadline; therefore the second arrived task will not be finished in time. The scheduling scheme shall be designed to avoid this type of confliction. The probabilistic scheduling algorithm CASA [26] relieved this issue by choosing the processing node in a probabilistic way. In CASA, n_i is chosen as the processing node with probability ρ_i, which is proportional to $\frac{1}{t(n_i)}$. Assume *n*3 and *n*2 are potential processing nodes, the source node still has the higher chance to schedule the current task to *n*3, but the probability that both source nodes schedule their current task to the same processing node is reduced to ${\rho_3}^2 + {\rho_2}^2$.

- Estimation of the bandwidth resource along the path

According to Eqn. (1), bandwidth plays an important role when calculate the estimated transmission time. Considering the situation that *s*0 offloads task to *n*3 and *s*1

offloads to $n4$, the transmission path from $n1$ to $n2$ is shared. No matter the transmission works in FIFO fashion or multiple access technique is used here, at least one of the transmissions has to be delayed. This delay may also cause the task not to be finished before deadline. Therefore instead of using fixed bandwidth information, the scheduler should adapt to the constantly changing bandwidth condition.

To summarize, without specifically considering an effective scheduling scheme for multiple sources and a method to get the context information, it is impossible to achieve intended improvement in local mobile clouds. The situation is even more complicated when the size of the local mobile cloud gets large. In this thesis, propose the adaptive probabilistic scheduler for local mobile clouds. It relies on the QoS OLSR routing scheme to provide periodic context information. It inherits the probabilistic scheduling algorithm but allows adaptive reconfiguration to improve performance under the unpredictable network condition.

CHAPTER 4 THE PROPOSED ADAPTIVE PROBABILISTIC TASK SCHEDULER

To solve the above task scheduling problem, we propose a distributive adaptive probabilistic scheduler which consists of two phases, namely the resource discovery phase and the adaptive probabilistic scheduling phase. In the resource discovery phase, source nodes are able to get the context information about the nearby processing nodes. In the adaptive probabilistic scheduling phase, the scheduler will choose one processing node to execute the task *j*. These two phases work together to ensure the performance in local mobile clouds gets improved.

A. Phase I: Resource discovery phase

The proposed resource discovery scheme is based on QoS OLSR [31]. There are two kinds of control messages carrying resource information:

- *Modified Hello Messages,* which are sent locally (i.e. broadcasted to one-hop neighbors) to enable a node to discover its local neighborhood (as HELLO messages in the QoS OLSR protocol [31]);
- *Modified Topology Control (TC) Message,* which are sent to the entire network through Multipoint Relay (MPR) nodes [32] to allow the distribution of the topology and context information to all the nodes (as TC messages in the QoS OLSR protocol [31]).

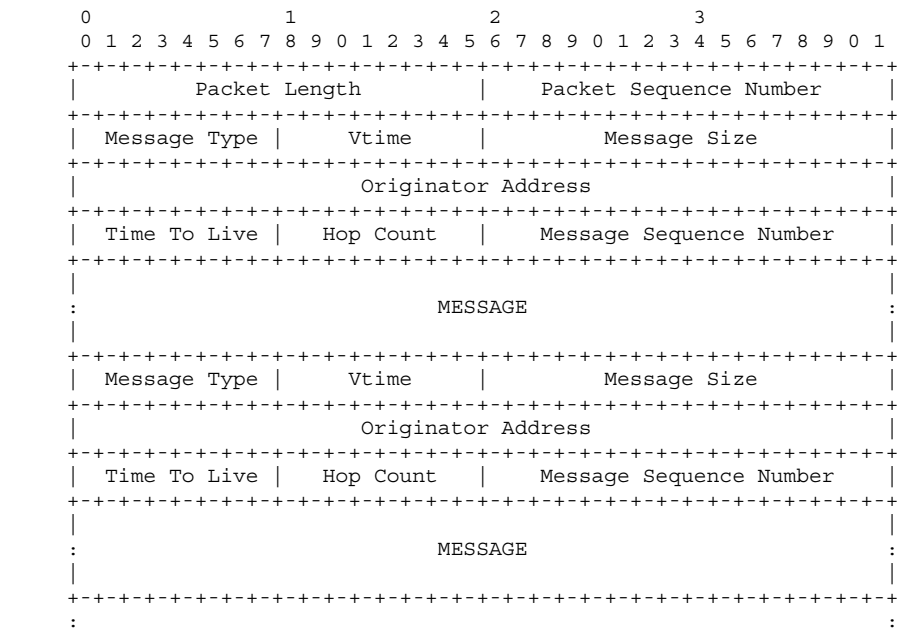
Note that both two types of messages are sent periodically. The emission interval should be a variable related to how fast the network changes. The faster the network changes, the shorter the emission interval should be. According to [32], the emission intervals for the original Hello massages and the original TC massages are 2s and 5s. The emission intervals of the Modified Hello Messages and the Modified TC massages are

going to be short considering the tight time constraint set by the applications. How the emission intervals affect the performance in local mobile cloud is to be evaluated in Chapter 5.

Our proposal is toward an extension of the routing table by adding the following parameters of each neighbor node into the two types of control messages:

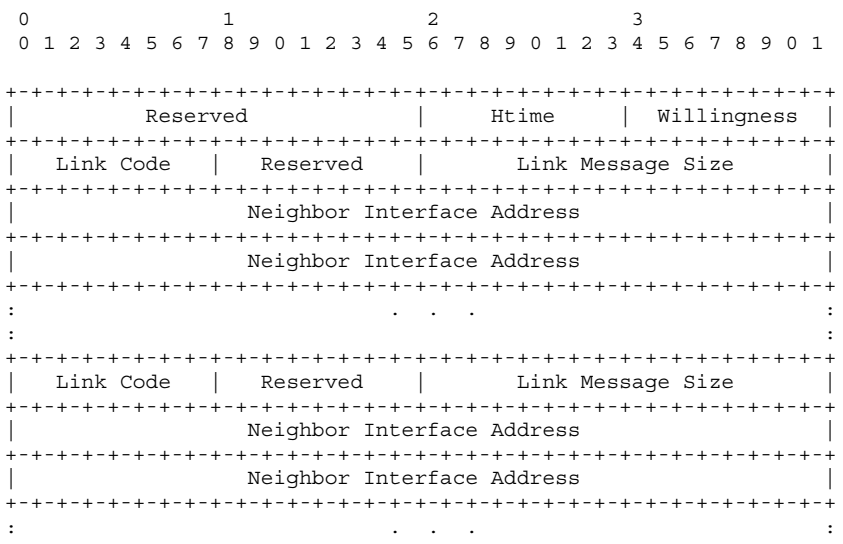
- Device parameter: indicate the processing speed $mips_u$ and energy consumption coefficient e_u of node u.
- Queue length: current queue time $t_{q,u}$ at node u.

The basic layout of any packet in QoS OLSR is as follows (omitting IP and UDP headers):



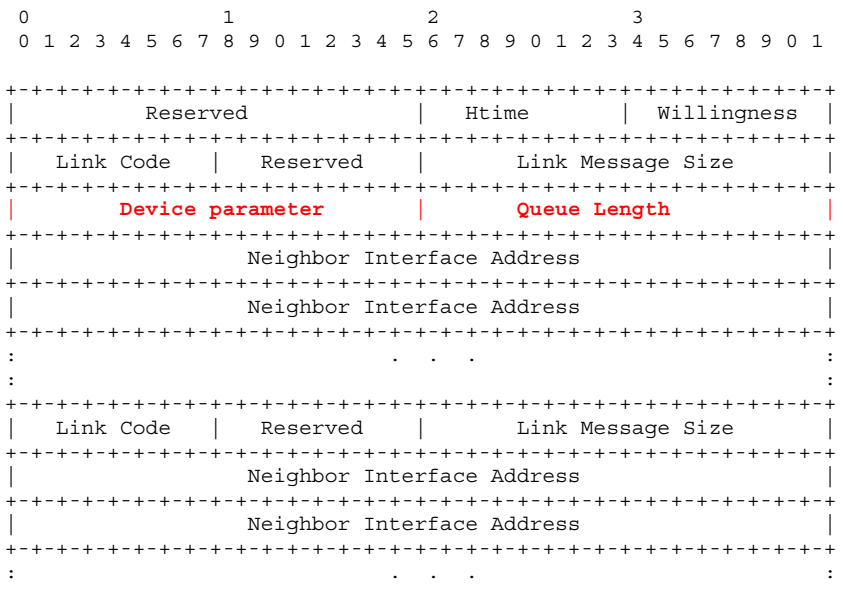
The description of each field can be found in [35]. The 'MESSAGE' field carries the control message. The difference between original control messages and modified control messages are given below:

Original Hello Message:



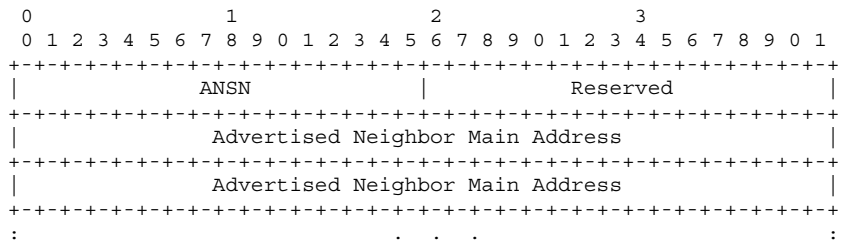
This is sent as the data-portion of the general packet format with the "Message Type" set to HELLO_MESSAGE and the Time to Live (TTL) field set to 1.

Modified Hello Message:



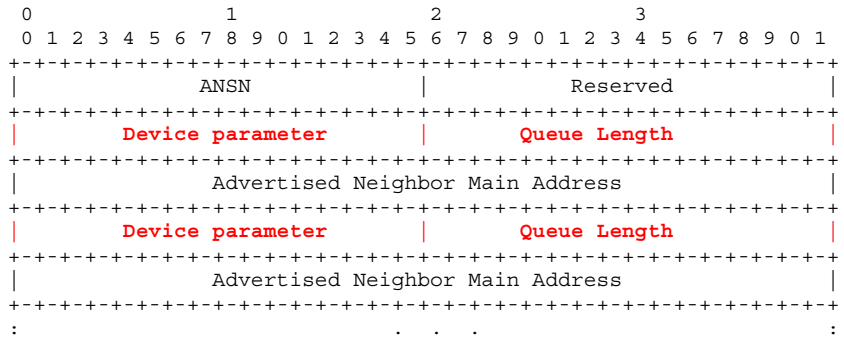
By adding device type and queue length, the current context information of the node can be updated at its neighboring nodes.

Original Topology Control (TC) Message:



This is sent as the data-portion of the general message format with the "Message Type" set to TC_MESSAGE. The TC Message can be diffused into the entire network. By varying the TTL value, the size of the network can be controlled. Only those processing nodes chosen to be MPR nodes will send and relay the TC message. 'Advertised Neighbor Main Address' field contains the main address of a neighbor node.

Modified Topology Control Message:



The modified TC Messages not only tell who its neighbors are but also carry its neighbors' context information. Having the modified control messages, each node can find other nodes' information by using a distributed algorithm shown in Figure 4. After executing the algorithm listed in Figure 4, all nodes in the network shall obtain a neighbor table in its local memory.

Resource Discovery Algorithm

Input: Control messages(Hello message or TC message)
Output: Neighbor table(The table storing the node parameter for each node)
Function: Handle control messages and update the neighbor table at node *u*
Procedure body:

```
{
initialize the neighbor table
listen control messages
if (Message Type== HELLO_MESSAGE)
{
        update the neighbor table
        if (node u is an MPR node)
            construct/update TC message
 }
if (Message Type == TC_MESSAGE)
{
        update neighbor table
        if (node u is an MPR node)
            forward the TC message to all of its neighbors
 }
}
```

Figure 4, Resource discovery algorithm

B. Phase II: Adaptive probabilistic scheduling phase

Each time a source node u receives a task j submitted by its local user, it estimates the energy consumption $Task\ Energy_{j,v}$ and the completion time $Task\ completion\ time_{j,v}$ of task j on every potential processing node $v \in P$. These two variables are obtained according to Eqn. (8) and Eqn.(12), respectively.

The scheduler keeps a set P' of processing nodes which satisfy Eqn. (19). The scheduler will randomly choose one processing node from the set. The probability that node $v \in P'$ is chosen to be the processing node is ρ_v.

$$Task\ completion\ time_{j,v} < T_j - t_{margin} \tag{19}$$

$$\rho_v = \frac{1}{Task\ Energy_{j\,j,v}} / \sum_{v \in C} \frac{1}{Task\ Energy_{j\,j,v}} \tag{20}$$

In the dynamic wireless environment, the tasks cannot be guaranteed to be completed before deadline. The reasons could be 1) there is still some chance that multiple source nodes schedule tasks to one processing at the same time; 2) the bandwidth of certain network connection is shared among multiple transmissions while Eqn. (11) gives an optimistic estimation of the transmission time; 3) unexpected network event, such as node's movement changes the routing path of the network or the processing node ran out of battery before finishing execution of the task.

When failure happens, the scheduler can adjust its t_{margin} value to avoid continuous future failures. When a_1 consecutive failed tasks occur, the time margin will be increased by Δt_1. After receiving a_2 consecutive successful tasks, the scheduler will lower the time margin by $\Delta \mathrm{t}_2$. That means when the network condition is degrading, the scheduler tends to choose the more powerful processing node at the cost of increasing energy consumption. The adaptive probabilistic scheduling algorithm running at each source node is given in Figure 5.

In this thesis, the proposed scheduling algorithm is compared with other scheduling schemes, including:

- Round robin scheduler: The scheduler assigns tasks in a round robin way. When the first task arrives, it is scheduled to the first node in P. It schedules the second task to the second node in P, and so on.

Adaptively Probabilistic Scheduling Algorithm
Input: (1) Neighbor table from Phase I, (2) task set *J*
Output: Scheduling (Mapping)
Function: Schedule the tasks to processing node in local mobile cloud
Procedure Body:
Set the number of consecutive successful tasks $n_s = 0$
Set the number of consecutive failed tasks $n_f = 0$
Task $j \in$ J arrives at source node u, record current time t_start_j.
P′ ← ∅
//Step 1: find eligible processing nodes
for each $v \in P$
{ $Estimated\ task\ completion\ time_{j,v} = t_{q,v} + t_{execution} + t_{transmition}$
$Task\ Energy_{j,v} = Computation\ Energy_{j,v} + Communication\ Energy_{j,v}$
if ($Estimated\ task\ completion\ time_{j,v} < T_{constraint} - t_{margin}$)
$P' = P' \cup v$
}
//Step 2: calculate probability that the eligible processing node being chosen as the processing node.
for each $v \in P'$
{ $\rho_v = \frac{\frac{1}{Task\ Energy_{j,v}}}{\sum_{v \in P'} \frac{1}{Task\ Energy_{j,v}}}$
}
Randomly select processing node w based on probability ρ_w
//Step 3: schedule j to processing node w and update estimated queue time on node w
Send j to w
$t_{q,w} = t_{q,w} + \frac{C_j}{mips_w}$
//Step 4:
Receive result of task *j* from the processing node *w,* record current time $t\_comlete_j$.
$Task\ completion\ time_j = t\_comlete_j - t\_start_j$
if ($Task\ completion\ time_j \leq T_j$){
n_s++;
$n_f = 0$;
}
else {
n_f++;
$n_s = 0$;
}
//Step 5: adaptively adjust t_{margin}
if ($n_f > a_1$) {
$t_{margin} = t_{margin} + \Delta t_1$;
$n_f = 0$;
}
If ($n_s > a_2$) {
$t_{margin} = t_{margin} - \Delta t_2$;
$n_s = 0$;
}

Figure 5, Adaptively probabilistic scheduling algorithm

- Greedy scheduler: The scheduler always sends the task to the most energy efficient node with the estimated completion time within the time constraint.
- Probabilistic scheduler: The scheduler keeps a set P' of the processing nodes satisfying $Task\ completion\ time_{j,v} < T_j$. The scheduler will randomly choose one processing node from the set P'. The probability that node $v \in P'$ is chosen to be processing node is calculated according to Eqn. (20).

Note that the round robin scheduler does not rely on the context information. As mentioned in Chapter 2, most existing greedy schedulers in local mobile cloud didn't consider the context information, for comparison purpose, we use the same proposed resource discovery scheme for the greedy scheduler. Probabilistic scheduler is based on CASA [26]. The difference between the probabilistic scheduler and the proposed adaptive probabilistic scheduler is the former does not have t_{margin}.The probabilistic scheduler does not adjust itself when failed tasks occur.

C. An illustrative example

We use Figure 6 as an example to demonstrate the difference between different schedulers. Related parameters are listed in Table 2.

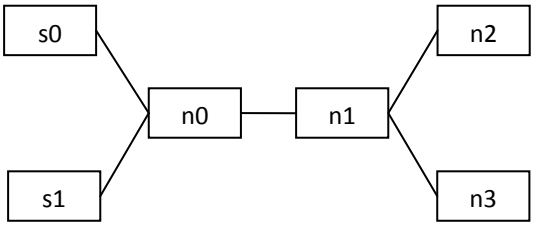


Figure 6, Sample local mobile cloud

Table 2, Parameter settings of the illustrative example

Parameter	Value
Task arrival intervals	0.5s
Task data size	1000B
Task computation amount	200 million instructions(MI)
Task time constraint	1s
$e_{n0}/e_{n1}/e_{n2}/e_{n3}$	0.25/0.16/0.09/0.04 J/MI
$mips_{n0}/mips_{n1}/mips_{n2}/mips_{n3}$	500/400/300/200 MI/s

At the start time t = 0, two source nodes $s0$ and $s1$ start receiving tasks. There is one task arriving at each source node every 0.5s. For the proposed adaptive probabilistic scheduler, t_{margin} is initialized to 0s. If one task is unsuccessfully scheduled t_{margin} is increased by 0.5s; otherwise, t_{margin} is decreased by 0.5s.

Since communication energy is very small comparing to computation energy, we ignore the communication energy in this example. We also ignore the transmission time for simplicity. The energy consumption for task at the processing nodes n0, n1, n2 and n3 is 50J, 32J, 18J and 8J according to Eqn. (4), respectively.

Figure 7 shows the results of four schedulers. Tasks from different source are represented in lines with different colors. Solid line is the execution time while dashed line is the waiting time at the processing node. The length of each line is the completion time of the task. If the duration of the line is longer than 1s, the task is failed. The completion rate and the energy per successful task are listed for each scheduler.

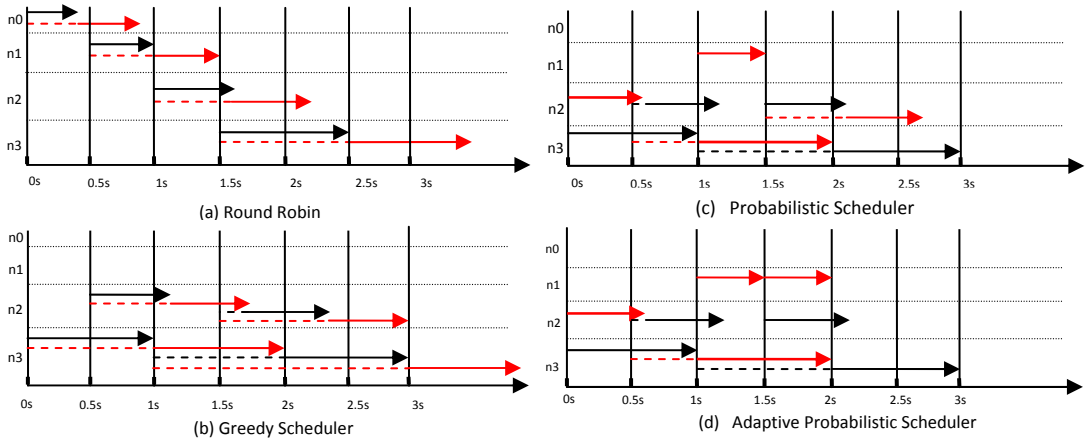


Figure 7, Results of the illustrative example

CHAPTER 5 NUMERICAL RESULT

To evaluate the performance of various scheduling algorithms in local mobile clouds, extensive simulations are conducted in OMNET++.

A. Simulation setup

The simulated local mobile cloud contains a group of nodes, each of which is capable of transmitting radio signals up to approximately 40 meters over an 11Mbps 802.11g wireless channel. Two network scenarios (with different node densities) are simulated. The first scenario, referred to as the small network, is created by randomly placing 10 nodes in a $200\text{m} \times 200\text{m}$ area. Among these 10 nodes, 2 nodes are source nodes and 8 nodes are processing nodes. The second scenario, referred to as the large network, is created by randomly placing 20 nodes in a $200\text{m} \times 200\text{m}$ area. Among these 20 nodes, 4 nodes are source nodes and 16 nodes are processing nodes.

There are two mobility patterns: the first one is the stationary network in which all nodes are stationary. The second one is the mobile network in which every mobile node $u \in V$ moves within the area according to the following pattern. It moves along a straight line for a certain period of time before it makes a turn. This moving period is a random number, normally distributed with average of 5 seconds and standard deviation of 0.1 second. When it makes a turn, the new direction (angle) in which it will move is a normally distributed random number with average equal to the previous direction and standard deviation of 30 degrees. Its speed is also a normally distributed random number ranging from 1 to 3 (m/s). A new such random number is picked as its speed when it makes a turn. All nodes can move to a random direction within the area with a speed uniformly distributed between 1 m/s and 3 m/s. All nodes have the same initial battery

level. The energy of a node is only consumed when the node processes a task or when the node transmits/receives a message, which can be calculated using Eqn. (6)-(8). The energy consumed in moving is ignored here.

Tasks are generated on each source node. The task arrival event is a Poisson process. Source nodes cannot be chosen to be the MPR nodes or processing nodes. Parameters are listed in Table 3.

Table 3, Parameter settings in simulation

Parameter	Value
Topology	Random
Network area	200m*200m
Network size	Small: 10 nodes /Large: 20 nodes
Communication range	Approximately 40 meters
Task data size	Varying from 1000 B to 8000 B
Task computation amount	Varying from 50 MI to 350 MI
Task time constraint	0.5s
Task arrival interval	Exponential distribution $\lambda = 0.2$s
Task arrival duration	200s
Computation ability of node u $mips_u$	Normal distribution in (1000,300) mips
Computation energy per MI of node u	$10^{-8} \times {mips_u}^2$ J
Maximum bandwidth	11Mbps

In the simulation experiments, the performance of the proposed task scheduling scheme was compared among different algorithms in terms of the following metrics.

- Completion rate: the ratio of the number of tasks finished within time constraint to the total number of tasks in Eqn.(14).
- Task waiting time: the duration between a task's arrival time at the processing node and its execution start time at the processing node.
- Average energy per successful task: the ratio of the total energy consumption to the number of successful tasks given in Eqn.(15).

In each experiment, 10 different random network topologies are generated. The simulation results are plotted using the average values derived from these 10 simulations.

B. Effect of varying Topology Control message interval

As introduced in Chapter 3, we modified the QoS OLSR by adding extra bytes to control messages. The influence of the modification is going to be evaluated in this section. In the original QoS OSLR, by default the Hello message interval is 2s and TC message interval is 5s.In the modified QoS OLSR scheme, the Hello message's interval is half of the TC message interval. The reason is that TC message is sent to the whole network with neighbor information updated at least once. Considering the tight time constraints (0.5s), we expect the control messages to be more frequently sent so the context information can be updated on time.

In order to gauge the overhead, we measure the number of bytes in the control messages, including HELLO messages and TC messages, in a small network. The *average overhead traffic* is defined as the number of bytes in the control messages transmitted per second in the whole network. The average overhead traffic shows how much network bandwidth is used for overhead messages. Figure 8 plots the average

overhead traffic of the original QoS OLSR and the proposed modified QoS OLSR vs. varying TC message interval.

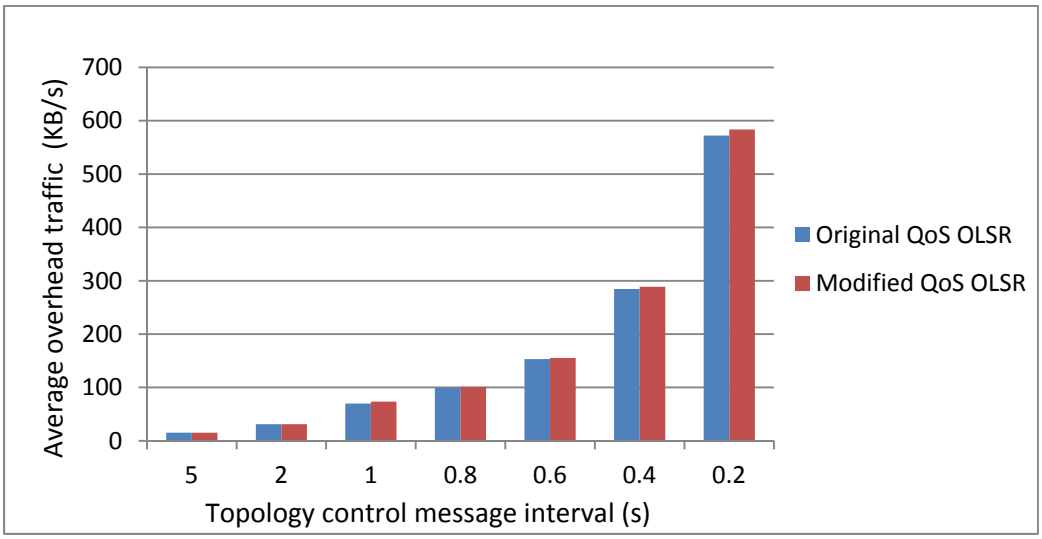


Figure 8, Average overhead traffic

The average overhead traffic increase almost exponentially as TC message get more frequent, when the TC messages interval reaches 0.2s, the traffic is nearly 600KB/s. The traffic of modified QoS OLSR increase only slightly comparing to the original QoS OLSR (up to 2%). The reason is that only 4 bytes are added to each Hello Message. In the TC message, though 4 bytes are added to each neighbor node, the neighbor list in TC messages is quite limited in a small network.

Next we investigate the performance of different schedulers under different TC message interval in a small network with 2 source nodes and 8 processing nodes. Figure 9 shows that except the round robin scheduler, the task completion rates of the other three schedulers first increase as the TC message interval gets shorter but decrease significantly

when the TC message interval reaches 0.2s. The reason is that with a long TC message interval, there are not enough control messages carrying context information, the schedulers are not able to get an accurate picture of nearby nodes. When control messages get more frequent, the context information is updated at the scheduler more frequently. As such, better scheduling results are obtained; the task completion rate becomes higher. However, when the overhead massages occupy large amount of network bandwidth (at TC interval of 0.2s), schedulers are unable to predict the accurate transmission time to the target processing node as in Eqn. (11). The scheduling results are significantly impacted, especially for greedy and probabilistic schedulers, which result in dramatic drop of the task completion rates.

The round robin scheduler does not rely on the control messages. It sends tasks to nearby nodes one after another. Thus its task completion rate remains the same. The proposed adaptive probabilistic scheduler has the highest completion rate for most cases. It has 16.6% and 3.6% higher completion rate than the greedy scheduler and the probabilistic scheduler, respectively. The reason is that with an increased t_{margin}, the scheduler will send tasks to a more computationally powerful node. Thus it has better chance to complete tasks before the time constraint. This result is further explained by Figure 10.

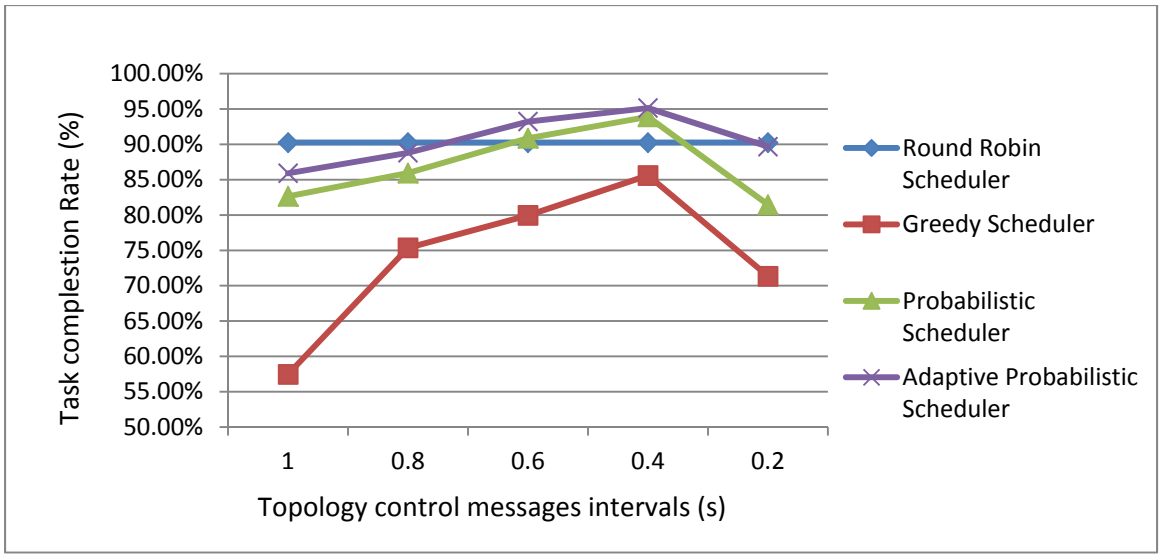


Figure 9, Task completion rate vs. TC message interval

In Figure 10, the simulation result shows the average waiting time per task at the processing node of the greedy scheduler is much higher than other schedulers. It is because both source nodes always tried to offload tasks to the most energy efficient node. A waiting queue is easily present in the targeting processing node and the long waiting time caused the task not to be able to finish before their deadline. The probabilistic scheduler and the adaptive probabilistic scheduler alleviate this problem by randomly choosing the processing node to reduce the chance that multiple tasks are offloaded to the same processing node. When confliction does happen, the latter has a better chance to offload to a more powerful node so that it has an even lower average waiting time, which is 2-4 times shorter than the greedy scheduler. The round robin scheduler has the shortest waiting time because it sends the tasks to nearby processing node evenly.

Figure 10 also demonstrated how TC message interval affects the scheduler's performance. With a shorter TC messages interval, more accurate context information is

achieved. Thus the schedulers are less possible to send the tasks to a node which already has long waiting queue.

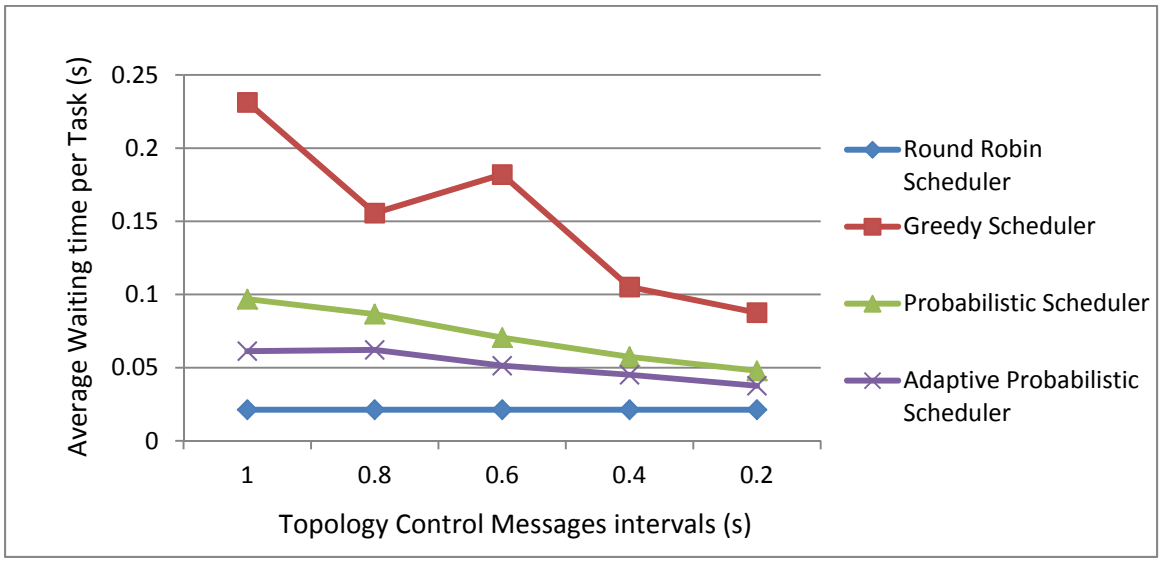


Figure 10, Average waiting time per task vs. TC message interval

Figure 11 shows that the average energy per successful task result is correlated with the task completion rate result. Except for round robin scheduler, the average energy per task of the other three schedulers first decreases as the TC message interval is reduced but increases significantly when the TC message interval reaches 0.2s. Among all four schedulers, the average energy per task of the adaptive probabilistic scheduler is the best. When the TC message interval is 0.4s, the adaptive probabilistic scheduler has the highest completion rate 95.14% and the lowest energy per successful task 1.60 J.

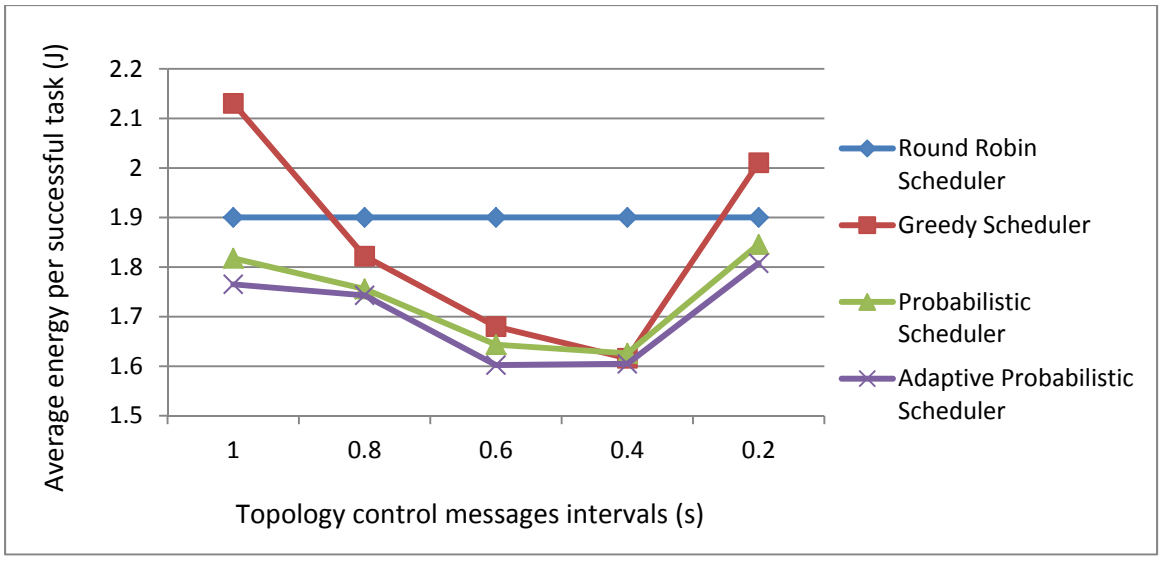


Figure 11, Average energy per successful task vs. TC message interval

C. Different size of network

The size of the network also affects the performance of the schedulers. In this experiment, the 0.4s TC message interval is adopted. A large network consists of 4 source nodes and 16 processing nodes. Figure 12 and Figure 13 show that in a large network the completion rate is lower but more energy can be saved. The task completion rate is 5% lower on average for all four schedulers in a large network. This is due to two reasons. The first reason is that the large network requires more hops between the source and the potential processing node. The increased hop count increases the unpredictability of the transmission time. The second reason is that 4 source nodes increase the workload of the network as more tasks need be scheduled at the same time. It makes the efficient scheduling even harder. This is confirmed by the fact that the greedy scheduler has an 8.73% lower task completion rate in large network while the probabilistic scheduler and

the adaptive probabilistic scheduler both only have about 5%. With 4 source nodes, there is a higher chance for the greedy schedulers to send tasks to the same processing node.

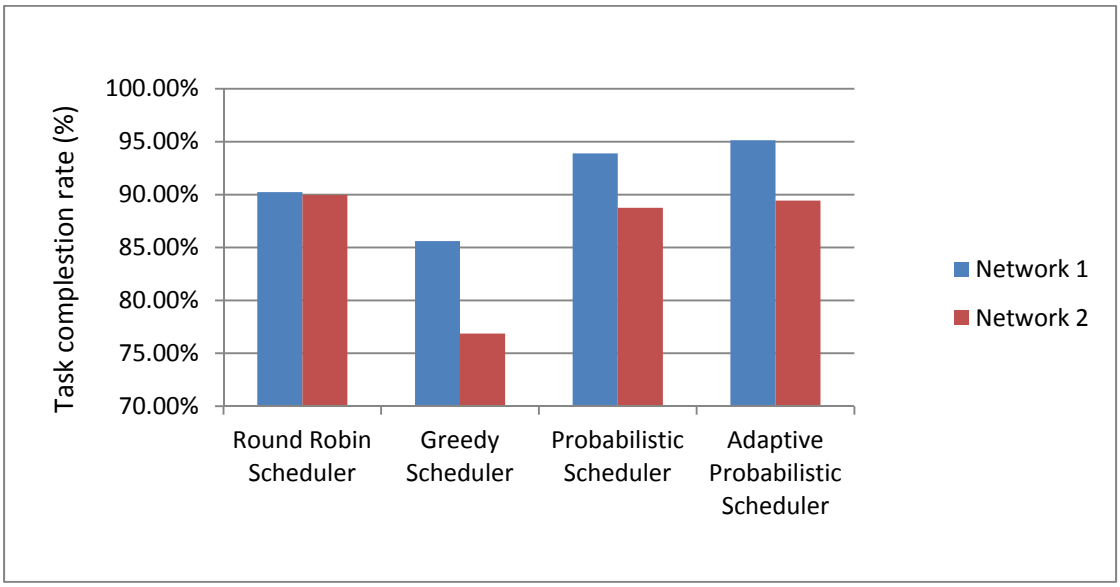


Figure 12, Task completion rate in small and large network

In Figure 13, the average energy per task is 12.5% higher on average for all four schedulers in the large network. The main reasons are greater transmission energy and decreased completion rate. The advantage of the adaptive probabilistic scheduler in large network is more significant. Comparing with other schedulers, the adaptive probabilistic scheduler reduced 29.3% energy per task on average in the large network while 10.9% in the small network. The reason is that there are more energy efficient nodes in the large network. Thus the adaptive probabilistic scheduler can adjust its parameter to choose a target processing node from a larger set of processing nodes to avoid confliction.

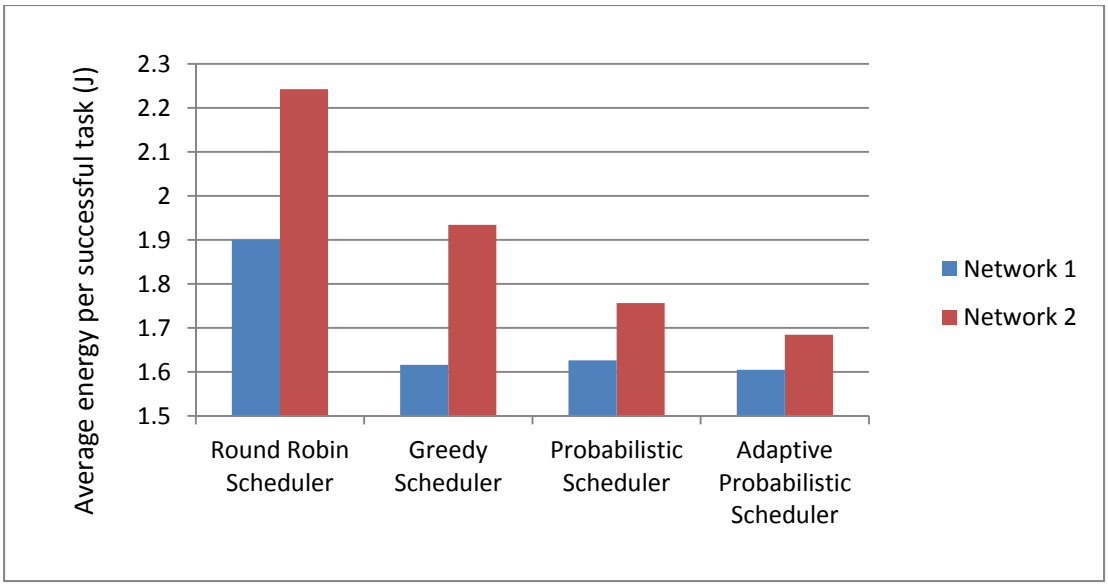


Figure 13, Average energy per successful task in small and large network

D. The impact of node mobility

This experiment explains how the performance is affected by the mobility of the network. Figure 14 shows that all four schedulers have a lower task completion rate in the mobile network than in the stationary network. Again, the adaptive probabilistic scheduler has the highest task completion rate in both stationary and mobile networks due to its ability to adjust to the changing context. The reason is mobility increase the context change.

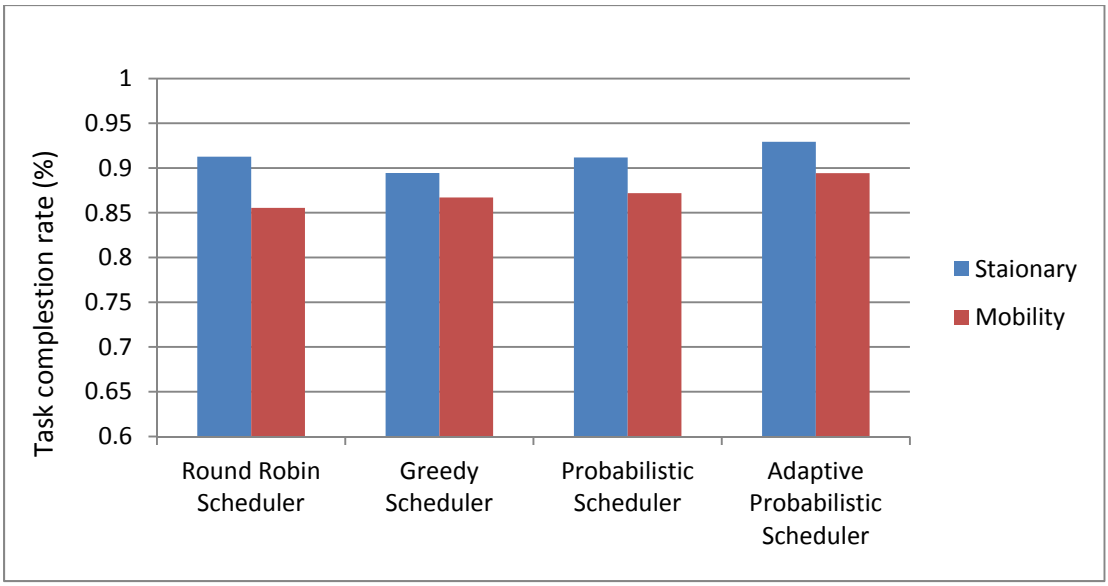


Figure 14, Task completion rate in stationary and mobile network

Similar situation happens to the average energy per task. The round robin scheduler consumes 1.4% more energy in the mobile network than in the stationary network. The greedy scheduler consumes 6.57% more. The probabilistic scheduler consumes almost the same amount of energy in both networks. The adaptive probabilistic scheduler consumed 1% less energy in the mobile network. This result shows that the average energy per task is not heavily affected by the node mobility as to the task completion rate.

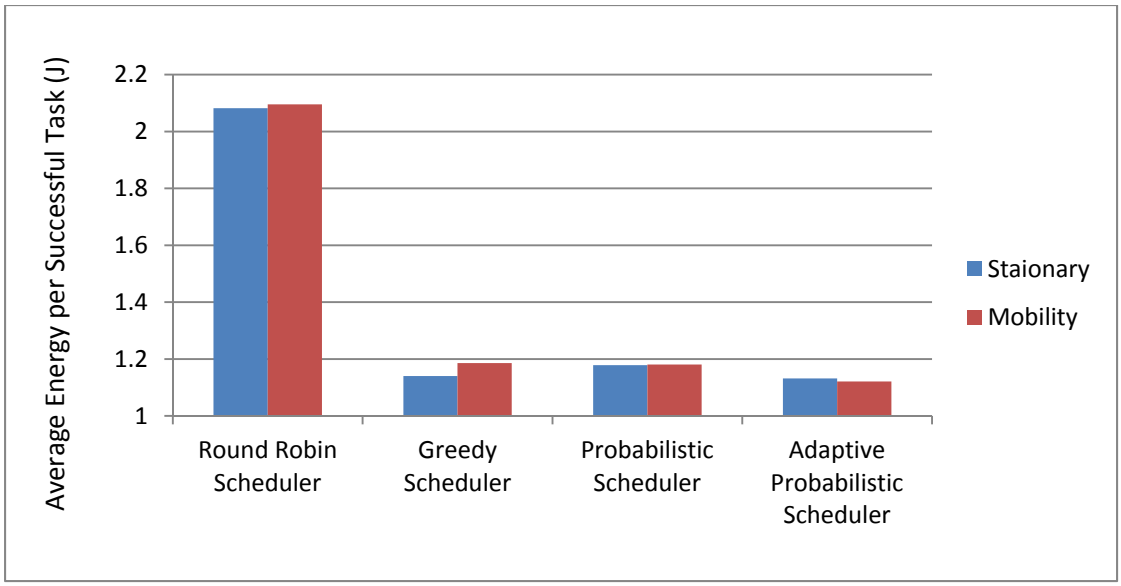


Figure 15, Average energy per successful task in stationary and mobile network

E. The impact of different task type

In this experiment, we investigate the performance of these four schedulers with different task type. First we fix the task data size then vary the computation amount. There are three workload scenarios: low, medium, and high workload. In low workload case, the task computation amount is uniformly distributed in [50..150] million instructions. In medium and high workload cases, the task computation amount in uniformly distributed in[150..250] million instructions and [250..350] million instructions, respectively. The average computation amount grows linearly from low workload to high workload.

Table 4, Three different workload scenarios

	Average computation amount (million Instructions)	Data size (bytes)
Low workload	100	1000
Medium workload	200	1000
High workload	300	1000

Figure 16 shows that the task completion rate drops with the increasing workload for all four schedulers. The task completion rate of the round robin scheduler drops abruptly under heavy workload. The reason is that the round robin scheduler does not check the context information. This impacts the scheduling results for tasks with heavy computation amount which require more computation time at processing nodes. The greedy scheduler drops linearly. The probabilistic scheduler and the adaptive probabilistic scheduler drop more slowly.

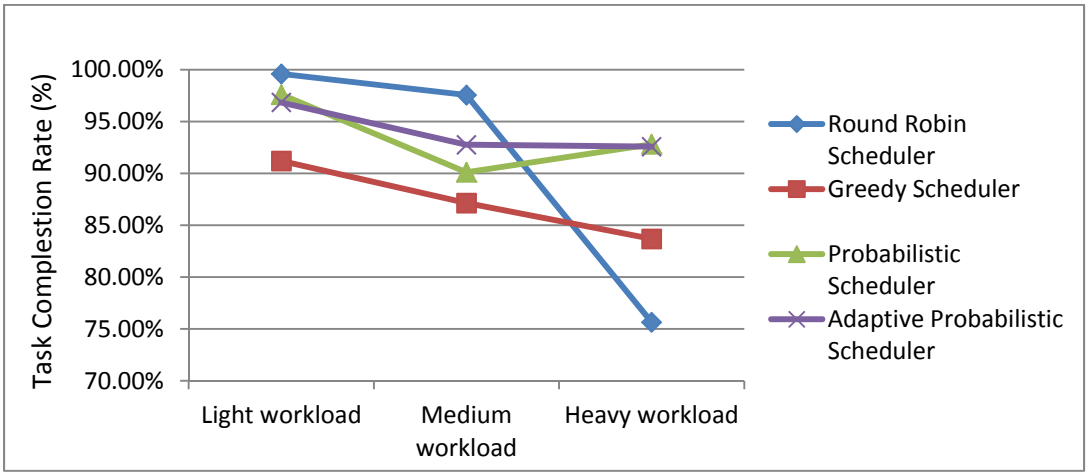


Figure 16, Task completion rate in different workload scenarios

Figure 17 shows that the average energy per successful task increases for all four schedulers . The average energy per successful task of the round robin scheduler in heavy workload scenario is 4.33 times as in low workload scenario. The average energy per successful task of the other three schedulers in heavy workload scenario is over 8 times as in low workload scenario. It is related to the square relation between the computational energy consumption and computation size in Eqn. (5).

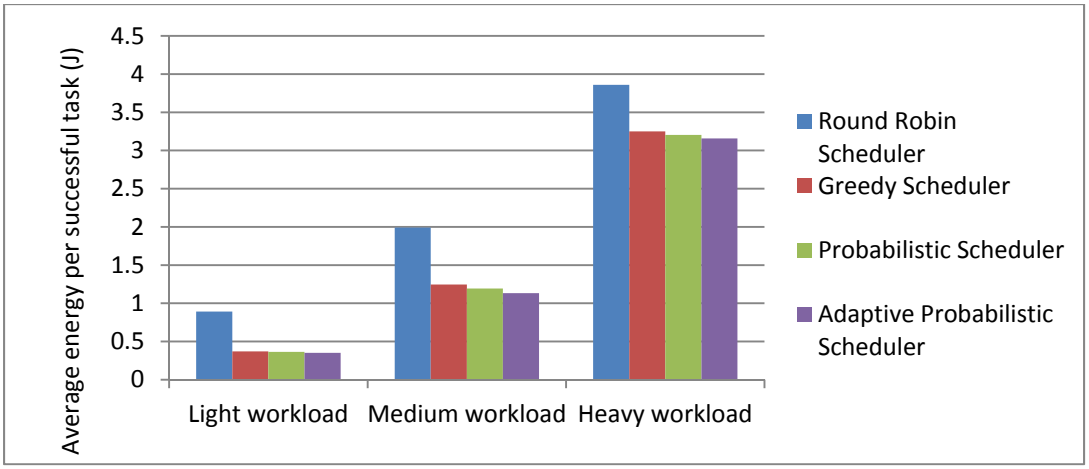


Figure 17, Average energy per successful task in different workload scenarios

Next we vary the task data size and fix the computation amount. There are three types of data size: small, medium, and large data size. In small data size case, the task data size is uniformly distributed in [500..1500] bytes. In medium and high workload cases, the task computation amount in uniformly distributed in [3500..4500] bytes and [7500..8500] bytes, respectively.

Table 5, Three different data size scenarios

	Average data size (bytes)	Average computation amount (million Instructions)
Small data	1000	100
Medium data	4000	100
High data	8000	100

Figure 18 shows that the task completion rate drops when data size increases. All four schedulers have the similar trend. The reason is with a larger data size, tasks need to wait longer time for other tasks finish transmitting because of the shared transmission medium. The inaccuracy of the estimated transmission time in Eqn. (11) increases. From the medium data size to the large data size, the task completion rate of the adaptive probabilistic scheduler only decreases by 9.8%. The task completion rate of the round robin scheduler, the greedy scheduler and the probabilistic scheduler decreases by 12.1%, 16.5% and 12.1%, respectively.

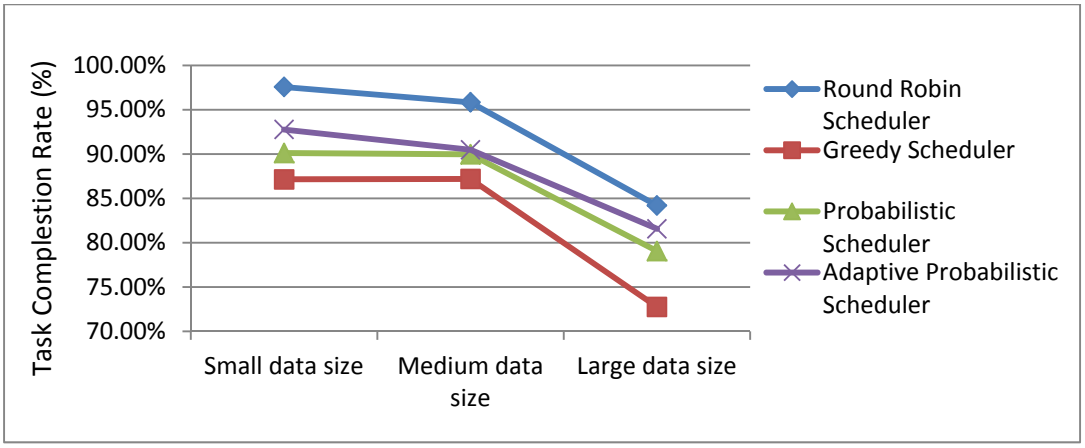


Figure 18, Task completion rate vs. different data size

Figure 19 shows that with increasing data size, the average energy per successful task of all four schedulers is also increased. This is attributed to the decreased task completion rate and the increased communication energy. From the small data size to the large data size, the average energy per successful task of the round robin scheduler is increased 13.87% from small data size to large data size. The greedy scheduler, the probabilistic scheduler and the adaptive probabilistic scheduler have increase of 20.53%, 23.07% and 25.66%, respectively. The adaptive probabilistic scheduler has the greatest energy increase because it tends to choose the more powerful processing nodes to improve the task completion rate when failed tasks occur, which explains why in Figure 18 the adaptive probabilistic scheduler has the smallest decrease in task completion rate.

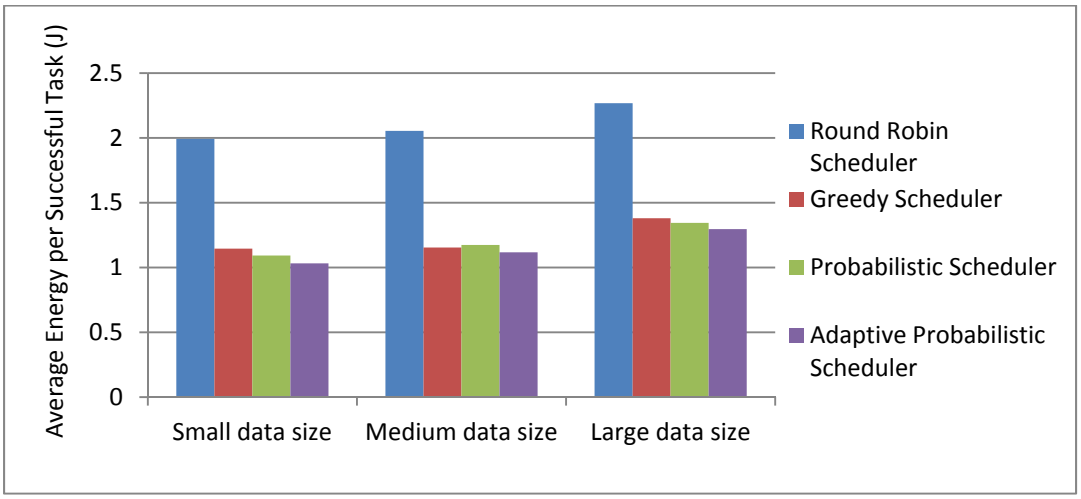


Figure 19, Average energy per successful task vs. different data size

F. Summary

First, the results in this chapter show how control message frequency affects the performance of four schedulers. Next the performance of four schedulers in local mobile

clouds with different network scenarios is evaluated. Then the influence of different task types on performance of four schedulers is investigated.

The round robin scheduler does not rely on the context information. It assigns tasks evenly to all nearby processing nodes which yield to a high completion rate as well as high energy consumption. When the computation workload increases, the completion rate of the round robin scheduler drops abruptly. By adjusting the frequency of the control messages which carry the context information, the improvement is obtained in the other three schedulers. The greedy scheduler suffers from low task completion rate because multiple tasks tend to be scheduled to the same processing node. This situation is even more obvious as the number of source nodes increases. The probabilistic scheduler provides higher task completion rate than the greedy scheduler. For most cases it also achieves a low average energy per task compared to the greedy scheduler. It reduces the possibility that more than one task is assigned to the same processing node. The adaptive probabilistic scheduler further improves the task completion rate by dynamically adjusting the set of potential processing nodes. It achieves highest task completion rate and the lowest average energy per successful task, which makes it an efficient task scheduler in local mobile clouds.

CHAPTER 6 CONCLUSION AND FUTURE WORK

A. Conclusion

In this thesis, we propose an adaptive probabilistic task scheduler for real-time applications in local mobile clouds. Based on QoS OLSR, source nodes receive periodical control messages to discover and update nearby resource information. The scheduler first estimates the task completion time and energy consumption at each potential processing node based on the context information collected through the modified QoS OLSR. Next, it schedules the current task to the proper processing node in a probabilistic way. Then it adaptively adjusts its time margin parameter to improve performance under the unpredictable network conditions.

Overall, the observed experimental results confirm that the adaptive probabilistic scheduler is able to reduce the average energy per successful task while maintain a high task completion rate. Furthermore, the performance benefit is more significant when the number of source nodes increases. In addition, the proposed scheduler can adjust itself to work for both stationary and mobile network scenarios. It also shows high adaptability with different task types. The adaptive probabilistic scheduler is a promising approach for real-time applications due to its scalability and flexibility in local mobile cloud.

B. Future work

Future work will include experiments using the more complex computation model at processing nodes. For example, each processing node may have multiple cores so that multiple tasks can be executed on one processing node simultaneously. How to utilize the parallel computation resources to further improve the scheduling performance is an interesting topic to be investigated.

Each task is only allowed to be scheduled once in the thesis. Future study could be explored to improve the performance of local mobile clouds by adding a rescheduling phase for tasks with high failure probability. By doing so, we could expect a higher task completion rate.

REFERENCE

[1] Gartner: the world's leading information technology research and advisory company. [Online]. http://www.gartner.com/

[2] Eduardo Cuervo et al., "MAUI: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 49-62.

[3] Yamini Nimmagadda, Karthik Kumar, Yung-Hsiang Lu, and C. S. George Lee, "Real-time Moving Object Recognition and Tracking Using Computation," in *The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 2449-2455.

[4] Roelof Kemp et al., "eyeDentify: Multimedia Cyber Foraging from a Smartphone," in *ISM '09 Proceedings of the 2009 11th IEEE International Symposium on Multimedia*, 2009, pp. 392-399.

[5] Tim Verbelen, Pieter Simoens, De Turck Filip, and Dhoedt Bart, "Cloudlets: Bringing the Cloud to the Mobile User," in *Proceedings of the third ACM workshop on Mobile cloud computing and services*, 2012, pp. 29-36.

[6] Romano Fantacci, Daniele Tarchi, and Andrea Tassi, "A novel routing algorithm for mobile pervasive computing," in *Global Telecommunication Conferrence*, 2010, pp. 1-5.

[7] Shu Shi, Cheng-Hsin Hsu, Klara Nahrstedt, and Roy Campbell, "Using graphics rendering contexts to enhance the real-time video coding for mobile cloud gaming," in *MM '11 Proceedings of the 19th ACM international conference on Multimedia*, 2011, pp. 103-112.

[8] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava, "A Survey of Computation Offloading for Mobile Systems," in *Mobile Networks and Applications*, 2013, pp. 129-140.

[9] Asaf Cidon, Tomer M London, Sachin Katti, Christos Kozyrakis, and Mendel Rosenblum, "MARS: adaptive remote execution for multi-threaded mobile devices," in *MobiHeld '11 Proceedings of the 3rd ACM SOSP Workshop on Networking, Systems, and Applications on Mobile Handhelds, Article No. 1*.

[10] E.E. Marinelli, "Hyrax: cloud computing on mobile devices using MapReduce," in *Masters Thesis, Carnegie Mellon University*, 2009.

[11] Suraj Sindia et al., "MobSched: Customizable Scheduler for Mobile Cloud Computing," in *45th Southeastern Symposium on System Theory*, 2013, pp. 129-134.

[12] Xiaoshuang Lu, Hossam Hassanein, and Selim Akl, "Energy aware dynamic task allocation in mobile ad hoc networks," in *International Conference on Wireless Networks Communications and Mobile Computing*, 2005, pp. 534-539.

[13] Omnet++ community site. [Online]. http://www.omnetpp.org/

[14] Amazon elastic compute cloud (EC2). [Online]. http://aws.amazon.com/ec2/

[15] Apple iCloud. [Online]. https://www.icloud.com/

[16] Apple Siri. [Online]. http://www.apple.com/ios/siri/

[17] Gonzalo Huerta-Canepa and Dongman Lee, "A virtual cloud computing provider for mobile devices," in *Proceeding MCS '10 Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, Article No. 6*, 2010.

[18] Emiliano Miluzzo, Ramón Cáceres, and Yih-Farn Chen, "Vision:mClouds – Computing on Clouds of Mobile Devices," in *MCS '12 Proceedings of the third ACM workshop on Mobile cloud computing and services*, June 2012, pp. 9-14.

[19] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14-23, October 2009.

[20] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *IMC '09 Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, 2009, pp. 280-293.

[21] Apache. Hadoop. [Online]. http://hadoop.apache.org/

[22] Gonzalo Huerta-Canepa and Dongman Lee, "A virtual cloud computing provider for mobile devices," in *MCS '10 Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, 2010.

[23] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti, "CloneCloud: elastic execution between mobile device and cloud," in *EuroSys '11 Proceedings of the sixth conference on Computer systems*, 2011, pp. 301-314.

[24] Tolga Soyata, Rajani Muraleedharan-Sreekumaridevi, Colin Funai, Minseok Kwon, and Wendi Heinzelman, "Cloud-Vision: Real-time Face Recognition Using a Mobile-Cloudlet-Cloud Acceleration Architecture," in *17th IEEE Symposium on Computers and Communications*, 2012, pp. 59-66.

[25] Karthik Kumar and Yung-Hsiang Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?," *Computer*, vol. 43, no. 4, pp. 51 - 56, April 2010.

[26] Ye Huang, Nik Bessies, Peter Norrington, Pierre Kuonen, and Beat Hirsbrunner, "Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 402-415, January 2011.

[27] Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, and Ramin Yahyapour, "Evaluation of Job-Scheduling Strategies for Grid Computing," in *GRID '00 Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, 2000, pp. 191-202.

[28] Vijay Subramani, Rajkumar Kettimuthu, Srividya Srinivasan, and P Sadayappan, "Distributed job scheduling on computational Grids using multiple simultaneous requests," in *Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing* , 2002, pp. 359 - 366.

[29] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *MobiCom '98 Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, 1998, pp. 85-97.

[30] Ahmed Helmy, "Efficient Resource Discovery in Wireless AdHoc Networks: Contacts Do Help," in *Resource Management in Wireless Networking*.: New York : Springer, 2005, vol. 16, pp. 419-471.

[31] Hakim Badis and Khaldoun Al Agha, "QOLSR, QoS routing for ad hoc wireless networks using OLSR," *Wiley European Transactions on Telecommunications*, vol. 15, no. 4, pp. 427-442, 2005.

[32] T Clausen and P Jacquet. (2003, October) Optimized link state routing protoco (OLSR). [Online]. http://www.ietf.org/rfc/rfc3626.txt

[33] Ahmad Afsahi Ryan E. Grant, "Power-Performance Efficiency of Asymmetric Multiprocessors for Multi-threaded Scientific Applications," in *IPDPS'06 Proceedings of the 20th international conference on Parallel and distributed processing*, 2006, pp. 25-29.

[34] Murali Annavaram, Ed Grochowski, and John Shen, "Mitigating Amdahl's Law through EPI Throttling," in *Proceedings of the 32nd annual international symposium on Computer Architecture*, 2005, pp. 298-309.

[35] P. Jacquet T. Clausen. (2003, October) Optimized Link State Routing Protocol (OLSR). [Online]. http://www.ietf.org/rfc/rfc3626.txt

[36] Christian Frank and Holger Karl, "Consistency challenges of service discovery in mobile ad hoc networks," in *MSWiM '04 Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, 2004, pp. 105-114.

[37] Stefanos Zachariadis, Cecilia Mascolo, and Wolfgang Emmerich, "The SATIN Component System-A Metamodel for Engineering Adaptable Mobile Systems," *IEEE Transactions on Software Engineering*, vol. 32, no. 11, pp. 910-927, November 2006.

[38] Trevor Mudge, "Power: A First-Class Architectural Design Constraint," *Computer*, vol. 34, no. 4, pp. 52-58, April 2001.

[39] C Lee, A Helal, N Desai, and B Arslan, "Konark: A System and Protocols for Device Independent, Peer-to-Peer Discovery and Delivery of Mobile Services," *the IEEE Transaction on Systems, Man and Cybernatics*, vol. 33, no. 6, pp. 682-696, November 2003.

[40] M. Reza Rahimi, Nalini Venkatasubramanian, Sharad Mehrotra, and Athanasios V. Vasilakos, "MAPCloud: Mobile Applications on an Elastic and Scalable 2-Tier Cloud Architecture," in *UCC '12 Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, 2012, pp. 83-90.

[41] Mahadev Satyanarayanan, "Mobile computing: the next decade," in *MCS '10 Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, Article No. 5*, 2010.

CV

Ting Shi

Degrees

2014 University of Nevada, Las Vegas (UNLV)
Master of Science (M.S.) in Electrical Engineering
Major in Electrical and Computer Engineering

2008 Tianjin University (TJU), Tianjin, China
Bachelor of Science (B.S.) in Engineering
Major in Electriconic Engineering

Honors and Awards

The Academic Success Center Employee Award Outstanding Graduate Assistant (2014)

Work Experience

2011~2013 Graduate Teaching Assistant, University of Nevada Las Vegas

2008~2010 3G Wireless Engineer, Huawei Technologies Co., Ltd

Thesis title

An Energy-Efficient, Time-Constrained Scheduling Scheme In Local Mobile Cloud

Thesis Examination Committee

Chair, Mei Yang, Ph.D
Committee member, Yingtao Jiang, Ph.D
Committee member, Ebrahim Saberinia, Ph.D
Graduate College Faculty Representative, Hui Zhao, Ph.D