Resource Monitoring and State Prediction for Mobile Cloud Platform

by

Xuesong Yang
B.Sc., University of Electronic Science and Technology of China, 2004
M.Sc., Shanghai Maritime University, 2007

A Master's Project Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

Resource Monitoring and State Prediction for Mobile Cloud Platform

by

Xuesong Yang
B.Sc., University of Electronic Science and Technology of China, 2004
M.Sc., Shanghai Maritime University, 2007

Supervisory Committee

Dr. Sudhakar Ganti, Supervisor
(Department of Computer Science)

Dr. Yvonne Coady, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Sudhakar Ganti, Supervisor
(Department of Computer Science)

Dr. Yvonne Coady, Departmental Member
(Department of Computer Science)

ABSTRACT

Mobile Cloud Computing (MCC) is a novel technology that combines mobile device with cloud computing. However, it has some inherent shortcomings compared with traditional cloud computing. For example, there will be unstable communications, intermittent connections, and limitation of power supply. Also unpredictably resources can join and leave cloud environment which makes mobile cloud operation and management more and more complex. In order to obtain stable resource participation of mobile devices, a combined push-pull method [11] has been proposed for collecting and analyzing dynamic information of mobile devices. At monitored nodes, a periodically push model is used to timely send the loading resources to the monitoring components. At monitoring nodes, a pull model is triggered by the change degree. If the change degree is greater than the threshold or the update loading information is missed during the time interval, the monitoring component immediately sends a query to monitored node for the latest loading information. The experimental results show that the model can effectively improve the monitoring performance for mobile environment and reduce communication overhead.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

The graduate study at the University of Victoria is a fascinating experience for me. It is a great pleasure to express my gratitude to many people who made this paper possible. I would like to thank:

Supervisor Sudhakar Ganti, for helping me through my graduate study. He always provides endless guidance, supervision and encouragement to me from the initial to the final level which helps me develop deeper understanding of the subject. I truly appreciate all the time and advice he gives to me throughout my time at University of Victoria.

My parents, my children and my wife, for supporting me during my studies. I deeply thank my parents for their timely encouragement and endless patience. I thank with love to my wife and my kids. My wife helps me get through this agonizing period in the most positive way, my children always bring cheerful laughter whenever I feel down. Thanks for making me strong. To all of you, thanks for always being there for me.

DEDICATION

I dedicate this project report to my parents who always support me.

Chapter 1

Introduction

Mobile Cloud Computing (MCC) is a new computing model that provides cloud computing environment to various mobile devices. Along with the rapid development of smart phone market and wireless communication environment, the communication and processing capabilities of smart phones are much more powerful than before. Especially, a large number of mobile devices can connect to each other via wireless and use "mobile as a service provider *(MaaSP)*", to generate a rich and powerful mobile cloud to provide pervasive computing, data collecting and processing services. Also, mobile devices can be considered as "mobile as a service consumer *(MaaSC)*", to improve the computation capability and energy efficiency of mobile devices by offloading computation tasks to the the cloud servers [10]. Mobile devices can also switch between the role of service providers and service consumers, offering or applying new service models for mobile cloud computing. With this powerful mobile cloud, mobile computing has been gaining chances as a feasible solution for large-scale problems.

By migrating local tasks to the mobile cloud, cost-effective and pervasive cloud services can be achieved, with a large number of mobile users and devices to work together in a distributed way [16]. However, compared to physically fixed cloud resources, mobile devices tend to provide a relatively inferior performance in terms of CPU, memory and storage capacity. They also have several problems, such as limited battery life, instability of wireless communications, and intermittent connections, etc.. It makes mobile cloud operation and management more and more complex. These problems make it difficult to use mobile devices as resources providers for effective stable job processing.

The resource monitoring is considered as a prerequisite condition and many major operations are based on monitoring data to provide users with service through

"pay-as-you-go" manner, such as network analysis, management, job scheduling, load balancing, event predicting, fault detecting, and fault recovery. Mobile Cloud Computing is more complicated than ordinary network owing to its heterogeneous and dynamic characteristics [11]. In this situation, accurate and timely resources monitoring activities are required to efficiently operate *MCC* platforms in order to manage the increasing complexity.

1.1 Project Overview

The concept of mobile Computing did not come into limelight until 2009, it has achieved great success in many fields since its birth, for example mobile commerce, mobile healthcare and mobile games. According to a report by Cisco [2]: traffic from mobile devices is anticipated to account for 60 percent of the total global IP traffic by 2016. The trend of *cloud computing, mobile computing and mobile cloud computing* can be seen from *Google search trends*, as shown in Figure 1.1.

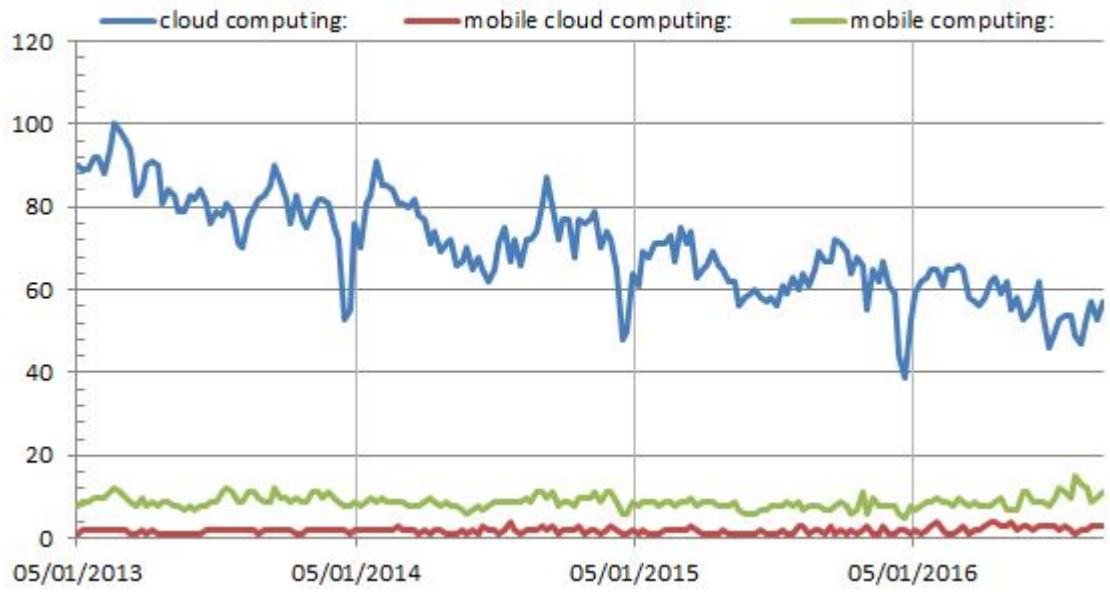

Figure 1.1: Compared CC, MC, and MCC in *Google Search Trends*

With the development of mobile applications and the support of cloud computing for a variety of services for mobile users, *mobile cloud computing (MCC)* is introduced as an integration of cloud computing into the mobile environment [8]. Mobile cloud computing brings new types of services and facilities to mobile users and takes full advantages of cloud computing. They are not only used for calls, SMS, emails and

web browsing, but also are considered as a necessary tool of life for navigation, fitness, and entertainment with the help of various of sensors, such as the light sensor, gravity sensor, gyroscope, accelerated sensor, and electronic compass. On the other hand, the explosive growth of *internet, 3G/4G, Wi-Fi, wireless broadband network* makes a variety of mobile devices easily accessible anytime, anywhere. It provides an important opportunity and the necessary infrastructure for mobile devices to access cloud computing. A variety of cloud-based mobile microblogging, mobile maps, mobile search, mobile payment, mobile networking games and other new applications are emerging and playing an increasingly important role. Therefore, mobile computing combined with cloud computing is one of the most important trends in the current computer field.

Mobile cloud architecture integrates the cloud computing with mobile environment. Figure 1.2 depicts the general architecture of *mobile cloud computing*. As we can see, mobile devices can access cloud services through mobile networks (telecom networks) or through access points.

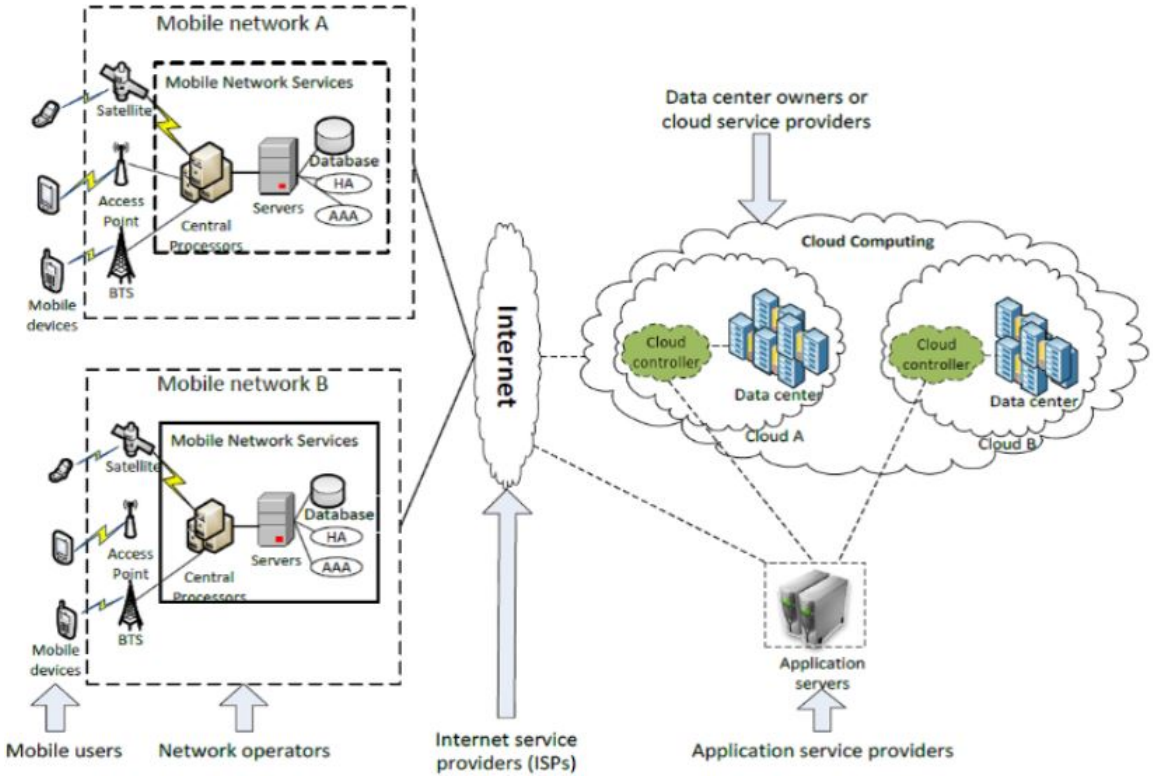

Figure 1.2: Mobile Cloud Computing (MCC) Architecture [7]

In the mobile network case, mobile devices are connected to mobile network

through a Base Station *(BS)* or via a satellite link. The telecom networks are further employed to connect to the Internet and provide cloud connectivity to the users. Therefore, if the users have mobile network connectivity, the users can access cloud based services through the Internet.

In the access point case, the mobile devices are connected to mobile network via access points that are further connected to the Internet service provider who provide Internet connectivity to the users. Therefore, the mobile cloud users can access cloud based services without utilizing telecom services, which may charge for data roaming. Moreover, *Wi-Fi* based connections provide low latency and consume less energy compared to *3G/4G* connections [6]. Consequently, mobile cloud users prefer to use *Wi-Fi* Internet connections whenever accessible.

1.2 Problems and Requirements

Despite the fast development of mobile hardware, it is still difficult to support computation intensive applications (e.g., image processing) on mobile devices. It hinders developers from bringing rich mobile application to mobile subscribers. Moreover, in the last years, the number of Cloud-based services have increased rapidly in mobile devices, and so is increased the complexity of the infrastructures behind these services [11]. To properly operate and manage such complex infrastructures effectively, efficient monitoring is constantly needed.

However, monitoring resources in mobile cloud computing is different from traditional cloud computing due to its heterogeneous and dynamic characteristics. If resource information is not provided correctly and timely, the incorrect information would cause accuracy problems. Therefore, a monitoring scheme that can accurately collect and analyse dynamic physical information for ensuring the stable estimation of resources is required. Furthermore, the monitoring data provided from mobile devices need to be deciphered as more accurate and sufficient state information to ensure *Quality of service (QoS)* of *MCC*. Both the provider and user can efficiently and effectively discover, reallocate or implement load balancing and task scheduling to adjust resources with the variations of resource usage.

In this project, I explore a resource monitoring method that combines push-pull between monitored and monitoring nodes. At monitored nodes, a periodically push model is used to timely send the loading results to the monitoring nodes. At monitoring node, a pull model is triggered by the change degree. It is the main algorithm for

resource monitoring that is used to predict the trends of resources on monitored node, and proactive identify and transfer key changes. The advantage of this method is to keep high consistency, only transfer the key changes to reduce the communication overhead, and implement dynamic frequency adjustment of data transmission.

1.3 Overview of The Report

The remainder of the report is organized as follows:

Chapter 2 gives the background and related work on different resource monitoring methods in cloud computing and *MCC* environment.

Chapter 3 presents the principle and details for combination of the push and pull methods, describes the technologies used in the development of the resource monitoring and also design details are given of the periodically push model periodicity and dynamically changing monitoring time interval of the pull model.

Chapter 4 shows the experimental results and the evaluation of the data presented above and the comparisons with the work of others, to show how much better the new approach is.

Chapter 5 draws the conclusion of the project. It also enumerates avenues of future work for further development of the concept and its applications.

Chapter 2

Related Work

The mobile cloud is based on the standard cloud service that provides the aggregation of computing as a utility and software as a service [18] where the applications are delivered as services over the Internet [1]. Nowadays, mobile cloud computing can be considered to deliver the service with *'on demand'* by mobile cloud service providers: software as a service *(SaaS)*, platform as a service *(PaaS)*, and infrastructure as a service *(IaaS)* [5].

2.1 Cloud Computing vs. Mobile Cloud Computing

The concept of mobile resource monitoring is used to exploit the computing and storage capabilities of the external mobile cloud and reduce intensive application to be executed on the local device. However, monitoring resources in a mobile is different from the same task in cloud and grid computing [9]. For instance, in mobile cloud computing, the network connectivity, amount of communication, bandwidth utilization cost, and mobile device energy are considered to be the foremost issues, which may not be the case (or least important) in cloud computing.

To the best of our knowledge, there is little research and analysis about platforms, techniques, and tools for mobile cloud monitoring infrastructures. Therefore, we present the differences between cloud and mobile cloud computing in terms of significant issues list in Table 2.1.

In order to solve inherent problems on mobile devices, resource monitoring is a task of paramount importance for both mobile resource Providers and Consumers.

Table 2.1: Cloud and Mobile Cloud Computing Comparison

Issues	Cloud Computing	Mobile Cloud Computing
Energy Consumption	✗	✓
Network Type	✗	✓
Bandwidth	✗	✓
Bandwidth Cost	✗	✓
Mobility	✗	✓
Context Awareness	✗	✓
Location Awareness	✗	✓
System Info.(CPU & Memory)	✓	✓
Security	✓	✓

On one hand, it is a key method for controlling and managing hardware and software infrastructures; on the other hand, it provides information and Key Performance Indicators (KPIs) for both platforms and applications. The continuous monitoring of the cloud and of its *SLAs* (for example, in terms of availability, delay, etc.) supplies both the providers and the consumers with information such as the workload generated by the latter and the performance that *QoS* offered through the cloud, also allowing to implement mechanisms to prevent or recover violations for both the provider and consumers.

2.2 *MCC* Resource Monitoring Model

In cloud computing, resource monitoring infrastructure consists of providers and consumers. Providers generate status report for monitored resources. Consumers make use of status information to migrate intensive computation to the cloud. Similarly, in mobile computing, consumers can obtain enhanced mobile experience and avoid constrained resources on mobile devices, allocate a specific set of mobile devices that can finish the tasks more accurately and efficiently according to monitoring information.

The monitoring services used in a cloud usually adopt *the pull model*, *the push model* or *the hybrid model* [21]. In the pull model, customers are responsible for "pull" information from resource providers to request resources information and status. However, in the push model, the provider "pushes" the new resource status to consumers. In other words, resource information is sent from the client according to the monitoring policy or trigger conditions on the server side. The push model is more accurate when predefined condition determines whether to trigger push operation or

not; while the pull model requires less transmission costs with proper inquiring interval. In the pull mode, the monitoring overhead is relatively small, because the resource information of a client is requested when it is needed, but this model has a long response time and is not widely used in dynamic environments because requests are sent to clients regardless of their states. In the push model, because a system administrator determines the monitoring time intervals, this interval is static. But, if the time interval of monitoring is very short, the overhead of collecting information increases. On the contrary, if the interval is very long, resource monitoring cannot keep correct state information in dynamic environments.

One of the previous researchers tried to determine the monitoring time interval and apply it through the observation of dynamic state of resource information [13]. It is based on the push model and has much less monitoring overhead than other models that monitor in real time. But this scheme only focuses on the CPU utilization in a high performance environment, so it is difficult to use this scheme in mobile grid environment where resources change more rapidly.

Schmidt et al. [17] proposed a solution to monitor Android for collaborative anomaly detection. The platform is based on *Linux OS level* that generates signals received by the actual monitoring components. *OS* level provides events that are recognized by the monitoring system. These events are initiated by kernel or file system changes. The *Linux application level* provides most of the functionality needed for monitoring and storing device and operating system information. There are two programs on Linux application layer, including *the monitoring application* and *the control daemon*. *The control daemon* is responsible for checking the status and persistence of the monitoring application. *The monitoring application* extracts information from the Linux kernel and the file system. On *Java application level* anomaly detection, detection collaboration, and detection response are realized where the corresponding states can be visualized in a user interface.

Yoon, Chanmin, et al. [20] tried to enhance *DevScope* system, an Android application that can probe operating system to obtain system resources and system configurations information, and generates a power model for Android smartphones. [20] developed an Android-based energy metering system that is called *AppScope*. *AppScope* employs *DevScope's* component power model and the power coefficient. Similarly [17], this solution monitors hardware's usage at the kernel level and accurately estimates energy consumption. *AppScope* is implemented as a kernel module and uses an event-driven monitoring method that generates low overhead and pro-

vides high accuracy. *AppScope* estimates energy consumption of each process based on a linear model that is based on *DevScope's* five core hardware components of smartphones, *CPU, display, cellular(3G), WiFi*, and *GPS*, and generates their power coefficients. The power coefficients will heavily impact the accuracy of energy consumption estimation for smartphone. This approach is suitable for the component power models that can be acquired in advance to balance the trade-off between benefits of the smartphones' performance and energy consumption.

2.3 Experiment Environment

Android is a very popular modern open source mobile platform, based on the Linux kernel and designed for mobile devices such as smartphones and tablets. In terms of software, Android has a very defined architecture. In Figure 2.1, each layer and its responsibilities of Android architecture structure [3] and process flow are displayed.

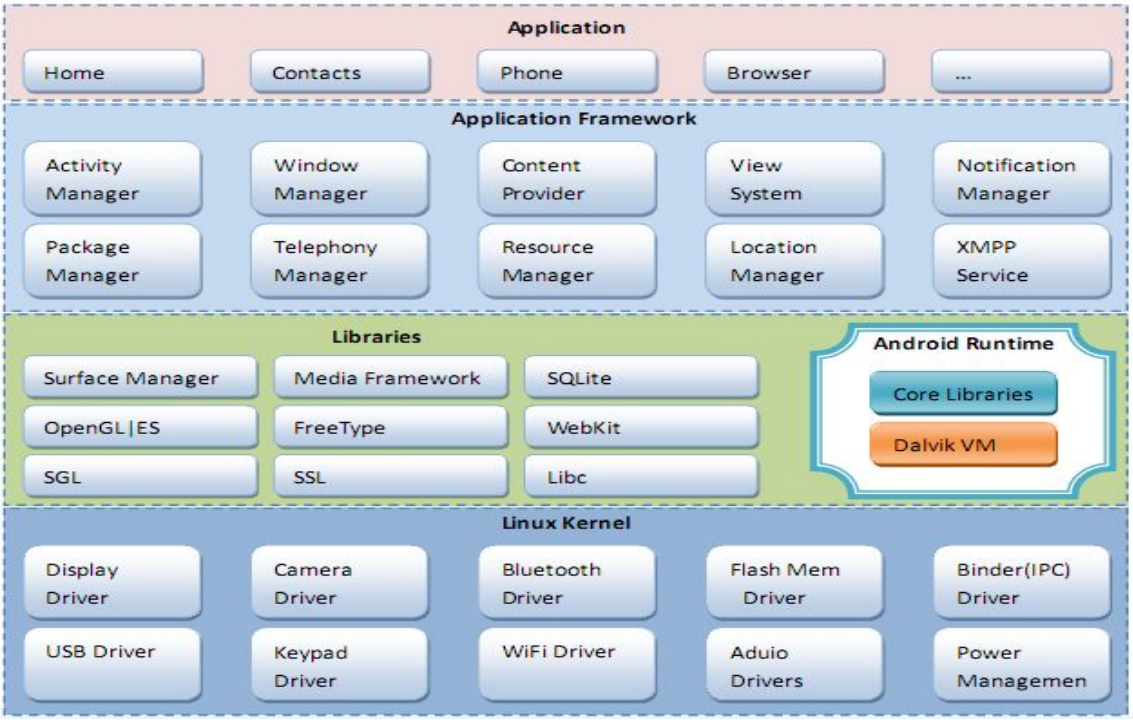

Figure 2.1: Android Architecture Structure and Process Flow[3]

Android executes a variety of applications via the *Dalvik Virtual Machine* where *Dalvik* is optimized to run on devices with constrained CPU, memory, and power resources. Although Android provides a shell interface, it lacks many of the abilities

of a typical Linux shell. We have to resort to other utilities, such as *Android Debug Bridge* or *BusyBox*.

2.3.1 *Android Debug Bridge (ADB)*

In order to obtain fine-grained monitoring information and adequate control for monitored devices, the powerful *ADB* tool should be pre-installed on each smartphone for experimentation. *ADB* is a versatile command line tool that can communicate with an emulator instance or connected Android-powered device. Based on *ADB*, we are not only able to send *Linux* shell command to the terminal, but also can use some specialty developer commands, for example the *dumpsys* tool.

2.3.2 *Dumpsys* Tool

Dumpsys is an android tool that runs on the device and dumps information about the status of system services. As part of *ADB*, Android Debug Bridge (ADB) provides a Unix shell which a variety of scripts can be easily executed. As part of ADB tools, dumpsys provides diagnostic function to output for all system services. The main benefits of using *dumpsys* is:

1. Easily get system information just in a single command line.

2. Flexibly dump CPU, RAM, Battery, storage stats, location information, for example:

```
#!/bin/bash
while true
do
    echo -n "TimeStamp:" >> /mnt/sdcard/sysinfo.log;
                    date +%s >> /mnt/sdcard/sysinfo.log
    dumpsys cpuinfo|grep "TOTAL">>/mnt/sdcard/sysinfo.log
    echo -e "\n" >> /mnt/sdcard/sysinfo.log
done
```

3. Conveniently implement periodical *push model* running a task via *Android Debug Bridge(ADB)*

2.3.3 *Task Scheduler*

In order to execute regular and repetitive monitoring jobs, Android provides support to schedule tasks. One of the *APIs* available to schedule background tasks is called *JobScheduler* that is located in the *android.app.job* namespace. It provides an interface for scheduling jobs that automatically and periodically execute routine tasks in the background. The task scheduler and dumpsys are executed at the user level. On each monitored node, *dumpsys* and *task scheduler* have been set as separate processes. The monitoring data is generated by dumpsys tool that collects CPU and memory usage, battery level, location, communication, device state, and network environments information. The task scheduler is assigned to timely run dumping system information and send monitoring results to the server.

There are different approaches to trigger scheduling recurring tasks in Android, such as

- Timer
- ScheduledThreadPoolExecutor
- BroadcastReciever with AlarmManager.

2.3.4 *Testbed*

The testbed for conducting the experiments to obtain mobile resources consists of three Android smartphones *(Samsung Galaxy Ace II)* and a *Samsung Galaxy Tab10.1 P7510*. The mobile phone OS had been updated to Android *v4.1.2 "Jelly Bean"* platform; and the tablet is running on the Android *v4.0 "Ice Cream Sandwich"*. Android provides an interface to system devices and services through a set of Java packages, including *android.os, android.hardware,and android.location*. This makes it easy to access and operate about system resource usage data, and location information.

A laptop plays role as the monitoring node that is connected with a 10M Ethernet connection. It can actively inquire nodes information and receives monitoring information from the nodes. Mobile phones and tablet are monitoring nodes, and are used to provide mobile resources for customer. The testbed hardware configuration is listed in table 2.2.

Table 2.2: Testbed Hardware Configuration Table

Component	**Galaxy Ace II**	**Galaxy Tab P7510**	**HP Pavilion G6**
OS Version	Android v4.1.2	Android v4.0	Windows 7 64-bit
CPU	Dual-core 800 MHz	Dual-core 1.0 GHz	A6-3400M 4*1.4GHz
Memory	1 GB RAM	768 MB RAM	8G DDR3 SDRAM
Storage	16 GB	4 GB	750 GB
Network	Wi-Fi 802.11n, 3G	Wi-Fi 802.11n	802.11n,10/100M
Location	GPS Assist	GPS Assist	N/A
Power rating	1500 mAh battery	7000 mAh battery	

2.4 Assumptions

This project depends on hardware and Android applications as described above with the following assumptions for a mobile cloud computing platform. We briefly explain why we made each assumption. Future work may allow these assumptions to be relaxed.

- Mobile users would honestly provide trusted services and accurate results when we try to use the mobile devices as cloud resources.
- The system will be built to capture the mobile resources information and the impact from security and privacy issue are not main concern.
- The processing time for generating, storing and *"pushing"* monitoring data is sufficiently short and thus can be ignored. Meanwhile, the monitoring nodes track the access of mobile resources in a real time and sequential way. The environment can be considered as a real-time mobile resource monitoring system.
- Android system has the ability to remove unnecessary and potentially insecure parts from the kernel. The resource monitoring system must have adequate access and control permissions that can monitor system information and sensitive resources.
- The Linux security layer provides user-based permission model, process isolation for security and piracy reasons [22]. The system allows monitoring node inquired message to trigger *"pull model"* and track the flow of system resources data.
- *ADB* and *dumpsys* are pre-installed with root permissions on smartphone. The script can access system setting and run programs as administrators in Windows, or running a command with *sudo* in Linux.

- The monitoring data collection must run passively in the background to prevent interference with current running applications.

Chapter 3

Design Methodology and Tools

Resource monitoring can be used for system management, network analysis, job scheduling, load balancing, event predicting, fault detecting, fault recovery and improve *QoS* for Mobile Cloud computing platform. As a first step of the MAPE (Monitoring, Analysis, Planning and Execution) loop [12], resource monitoring plays a key factor to ensure the efficient operation of applications. It has attracted many researchers and businesses' attention. Some popular monitoring tools have been widely used in large scale grid or cloud computing environments. These solutions are oriented to optimize the performance and availability of IT infrastructures. A few monitoring tools are described in [4].

In contrast to these state-of-the-art approaches that mostly focus on specific issues in distribution system or cloud monitoring, we cannot directly apply the monitoring models from cloud platform to mobile cloud computing due to the different features between cloud computing and *MCC*. However we can introduce this concept into mobile cloud computing with modification.

The ideal mobile resources monitoring platform can provide accurate data collection for system usage and performance, visualizing data at the same time. If the monitoring system cannot quickly identify unstable state or key change degree, even if it has a good computation offloading techniques and efficient program code, there is no guarantee for the performance of the application. To design this monitoring system, a dynamic, sequence-based system must be implemented to monitor the various resources and store the resulting data. To design a dynamic real time resources monitoring system there are four things to consider:

- How to trade-off between efficiency, accuracy and overhead in the resource mon-

itoring models;

- How to identify the key change degree for the load of monitored resource, and how to ignore non-critical changes, and reduce network load;
- How to define the monitored state information based on individual monitoring parameters, comprehensive analysis and evaluation of system status rather than isolated element information;
- How to employ an appropriate method to predict the state transition.

The later sections of this chapter cover how each of these requirements are satisfied. First, an overall system design analysis is discussed to assess the pros and cons of each design concept. Secondly, a high level overall view of the chosen design is explained. Later each module of the overall system design discussed in details.

3.1 Overall System Design

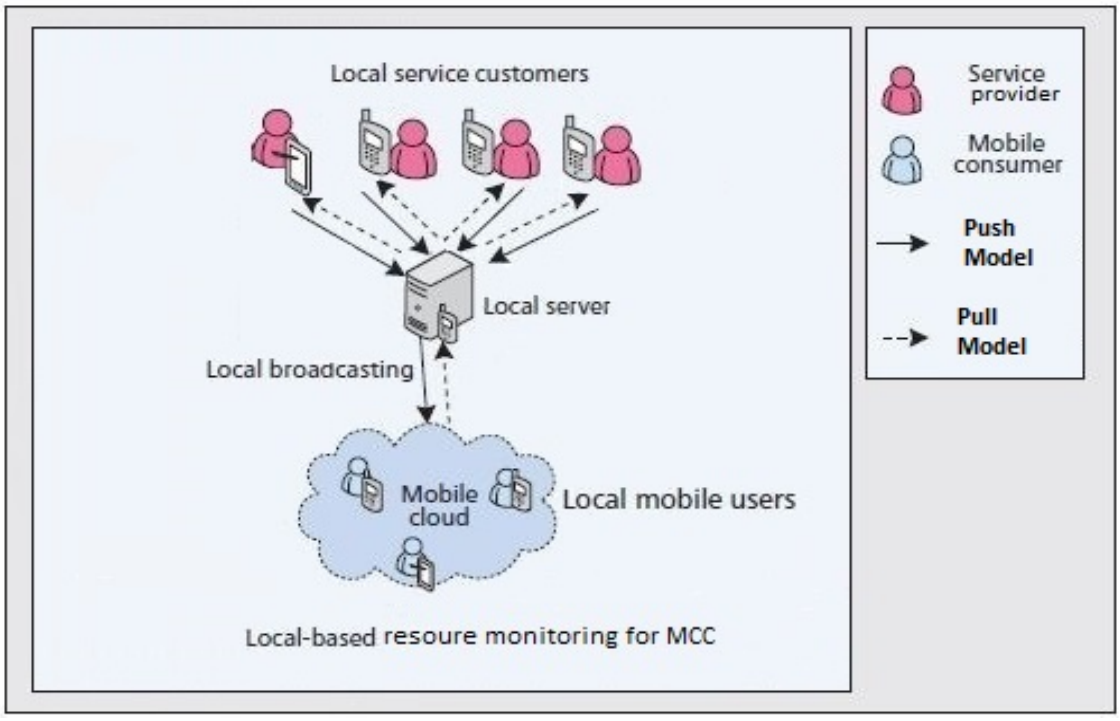

Figure 3.1: Overall Local-based Resource Monitoring System Design

The resource monitoring system design is based on the Android operating system. As shown in Figure 3.1, the design is divided into three major implementations: the Android devices implementation, the server side implementation and the mobile resource service implementation. We focus on the resource monitor rather than the service and the service module will not be discussed here. According to different data transmission mechanisms in this project, the Android devices implementation is further divided into two different modules: the push module and the pull module. The server side is to handle the collected data and make future state analysis based on client and server modules. The mobile users play roles both as potential service providers or as consumers in a local network. The related concept has been defined as follows:

1. Mobile Consumer: It refers to the local users who use the require cloud services through mobile cloud. They utilize the cloud services by outsourcing tasks to other mobile devices;

2. Service Provider: Mobile users with mobile devices can autonomously form a mobile cloud to provide cloud services, for online service consumers via cellular/WiFi networks, or connect with Bluetooth/NFC techniques. When a mobile user participates in an outsourced task, it can adopt local computing or require mobile cloud computing to execute this task;

3. Local servers: are generally equipped with dedicated mobile local gateways to disseminate the task information to neighboring mobile users and collect user report results. In addition, they can actively query resources information from resources.

3.1.1 Data Transmission Model Analysis

The data transmission model is the most important and crucial consideration of this thesis investigation. For the data transmission model design, the main requirement is that the model must be able to collect and transmit the communications of underlying resources in real-time. The monitoring services used in mobile computing usually adopt three basic methods between monitoring and monitored nodes: the pull model, the push model or the hybrid model. Each has its own advantages and disadvantages.

Push model

In the push model, as to mobile cloud computing, each monitored node is actively to send its resource information to the monitoring component (local server) without being foretold. The principle is shown in Figure 3.2 on the left side. Since a system administrator can easily determine the monitoring time intervals, this interval is static. But, if the time interval of monitoring is very short, the overhead of collecting information increases and further aggravates network load. On the contrary, if the interval is very long, resource monitoring cannot keep correct state information in dynamic environments. Especially, when some emergency happens, it would not send it immediately to the master node, that is what we do not expect to see.

One of the solutions is to select proper threshold for keeping maximum consistency between service provider and local server. However, if threshold is small, minor changes result in too much information transmission, which may place strain on the network. If the threshold is large, important updating may be lost.

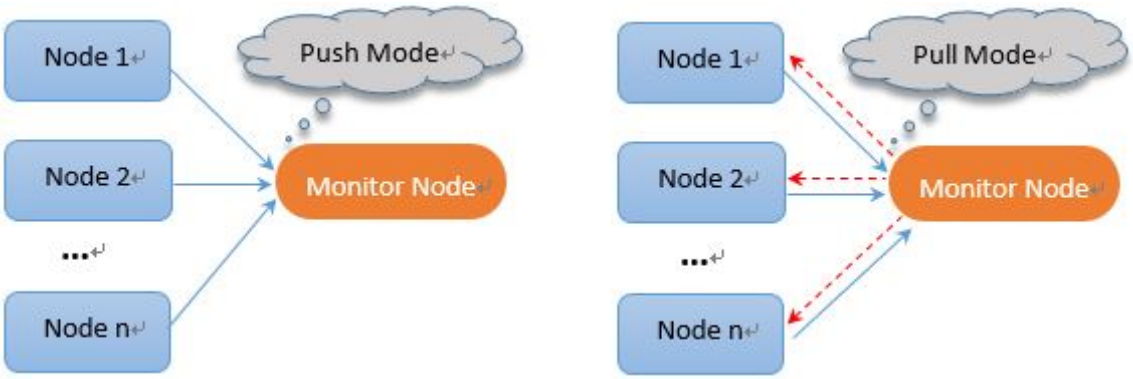

Figure 3.2: The Principle of the Push and Pull Model

Pull model

In the pull model, the nodes of service provider are in a passive mode. They need the local server to tell them the time that they have to send the monitoring data. It is shown in Figure 3.2 on the right side. Because the request of resource information is required, the monitoring overhead gets less. But It is very important to choose the time interval in this model: too frequent pull aggravates network load; too long an interval cannot assure the freshness of the data, and may lose to capture information during the pull message response time. Meanwhile, the transmission of pull request

consumes time and bandwidth, also requests are sent to service provider regardless of their states.

In view of all the above, push model has high consistency but low efficiency, while pull model has low consistency but high efficiency [10]. Based on the complementary properties of these two models, we adopt a periodical push and state-driven pull model for mobile resource monitoring, which employs both of the above two models, and automatically triggers pull model once the extent of change has been detected between the two adjacent states on local server [15].

In order to avoid complicated pull algorithm being applied to the limited resources mobile devices, we move pull model to the server side where it will be activated by the predefined threshold. The benefit of this method is to enhance effective monitoring without causing additional overhead and avoid the monitored nodes unavailability for a long period.

3.1.2 Server Module Analysis

The server module can be seen as a mobile computing platform for service consumers. It can provide trusted services for resource allocation, task processing and scheduled jobs for service providers or mobile users.

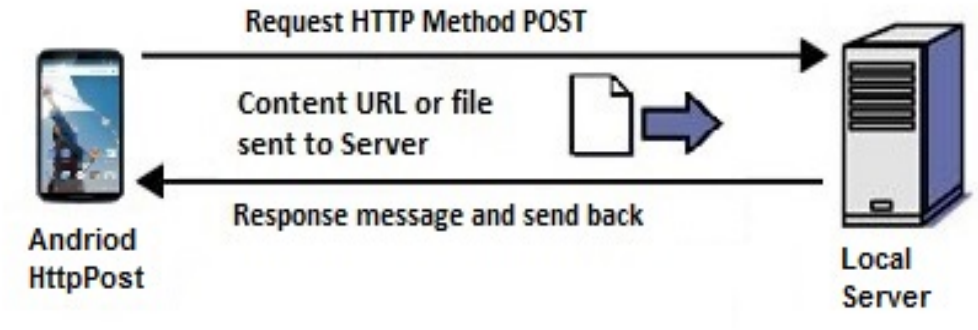

Figure 3.3: Overview of Server Module Design

In this project, the server side is to handle the collected data from service providers and make further state analysis based on exponential smoothing prediction on model. The server module is responsible for receiving monitoring data, processing it, and trigger pull model requests for accurate resource information. Specifically, the web layer handles incoming server communication. The application layer is responsible

to trigger pull algorithm for state prediction. Figure 3.3 explains the overall server module components and process flow.

The communication method between the mobile devices and the server is HTTP protocol that supports two different request methods: GET and POST. Due to security reasons and variable length of the monitoring data being collected, the HTTP POST Request method was chosen for communication between the mobile devices and local server. The tasks of the server module are to receive HTTP POST Requests and response to resource allocation. Thus, an Apache web service had to be installed to handle incoming HTTP Request. The web service is composed of the web layer and the application layer. The web layer receives HTTP POST Request, then the request is passed on to the application layer and a response message is sent back to mobile devices.

Exponentially Weighted Moving Average (EWMA) is adopted to predict the next state and calculate a monitoring time interval on the server side [14]. The purpose is to reduce the usage of system resources and make sure all the data reflects the real behavior of the system.

3.1.3 Data Collection Mechanism Analysis

In order to implement the data collection, the design is divided into two major parts: the Android devices implementation and the server side implementation.

The Android devices implementation is achieved by Android debug bridge (ADB), dumpsys tool and the Android logging system. These tools are used to dump system information, capture the monitoring logs and then send data to the server module. A scheduled job is established for periodically driven dumpsys instructions and shell scripts. The shell is not part of Android kernel, but uses the system kernel to execute programs [22]. Further details are based on the Android architecture structure and process flow shown in Figure 2.1. This facility allows for the logging of applications and system components. Since ADB dumps from Android devices to actively collect and sends information, the collection process is able to run passively in the background.

On the sever side implementation, the design was a simple one. The goal was to securely receive the data and establish an interface between the server and monitoring nodes. The communication protocol that is used is HTTP POST request. Essentially, it is the messenger. This is done though the use of Android HttpClient

prediction. First the HttpClient is initialized. Then the server implements an exponential smoothing prediction model model that is based on the closest time stamp to predict the next state. If the state changes from stable to unstable or disable, the latest resource information request is added to the POST request. Finally, the HttpClient forwards the POST request to the specific mobile device for further resource information. If the POST was successful, HttpResponse then receives a response message.

In summary, the system can passively retrieve resource information, send push result and trigger pull model. It allows the communication between the data collection module and server to carry on seamlessly. The next section will discuss the implementation details of data transmission model.

3.2 Data Transmission Model Implementation

Push mode has high consistency but communications overhead that will deteriorate if we frequently push data while pull mode has small communications overhead but cannot assure the accurate monitoring if long time interval is implemented. In order to ensure real-time synchronization of monitoring information and keep the stability and reliability of monitoring system, this project combined the advantage of the push and pull models.

The implementation consists of two methods: push and pull algorithm that have been implemented on monitored node and server side, respectively.

3.2.1 The Change Degree

On monitored nodes, load change is divided into key and non-key changes according to the change degree of monitored resources, only the key changes can be transferred since it can accurately reflect the monitored resource state, and ignore non-key changes to reduce unnecessary communication overhead.

On the monitoring side, if the servers cannot receive monitoring data in the communication interval, or the prediction state has been identified as disable or unstable, an active pull inquiry is activated in order to query the mobile devices performance.

The change degree is defined as the absolute value of the difference between the current status of a monitored resource and the status preserved in the corresponding

monitoring nodes. It is described in 3.1. Every meta data contains a time stamp which records the time at which a monitored resource collects status information [11]. Because each node acts as a relatively independent resource, there is no requirement for synchronization between the nodes.

$$Change\_degree = \frac{P(t_p) - C(t_c)}{MAX - MIN} \leq threshold(t_p \geq t_c) \tag{3.1}$$

The t_p represents the latest time stamp on the monitored node, and similarly, t_c represents the closest time stamp prior to time t on the monitoring node. We have $t_p \geq t_c$ because a monitored node's update is always prior to a monitoring nodes update. Therefore, $P(t_p)$ can be looked as the real status of the mobile device at time t and $C(t_c)$ denotes the status information that the monitoring node holds at time t. In fact, $C(t_c)$ represents the last update that the monitoring node receives. MAX and MIN are the maximum and minimum possible value of status.

For each monitoring pair, the most appropriate monitoring model depends on the setting of threshold. There are two conditions where the change will be looked as the key change, the two possible conditions are shown in (3.2).

$$\begin{matrix} monitoring \\ strategy \end{matrix} = \begin{cases} The\_change \geqslant the\_setting\_value\_of\_threshold \\ \\ After\_interval\_period, the\_change \geqslant min\_threshold\_value \end{cases} \tag{3.2}$$

According to the definition, when the load change is greater than the threshold, the monitoring data is transferred between monitored resources and monitoring component. When the load change is less than the threshold, the change is ignored to reduce transmission times and network load. Meanwhile, in order to keep high consistency, the time interval is set to identify load change during the monitoring period. Only when the change is greater than *min_threshold*, the monitoring data is transferred between monitored resources and monitoring component, the purpose is to ensure continuous communication between monitored resources and server, also to avoid useless data transmission.

An example is used to illustrate key and non-key changes below. Assume the CPU usage is considered as a monitoring resource, the threshold value is set to 0.2, *min_threshold* value is set to 0.04 and the interval is set to 5s. The change of degree is shown in table 3.1.

Table 3.1: An Example for Key and Non-key Changes

Time Stamp	$Cpu\_P(t_p)$	Change	Timer	$Cpu\_C(t_c)$	threshold	*min_threshold*	*interval default 5s*	Is key Change
1410278022	14	0	0	14	0.2	0.04	5	No
1410278023	13	0.01	1	14	0.2	0.04	5	No
1410278024	6.2	0.078	2	14	0.2	0.04	5	No
1410278025	3.5	0.105	3	14	0.2	0.04	5	No
1410278026	48	0.34	0	48	0.2	0.04	5	Yes
1410278027	2.7	0.453	0	2.7	0.2	0.04	5	Yes
1410278028	14	0.113	1	2.7	0.2	0.04	5	No
1410278029	2.2	0.005	2	2.7	0.2	0.04	5	No
1410278030	3.4	0.007	3	2.7	0.2	0.04	5	No
1410278031	13	0.103	4	13	0.2	0.04	5	Yes
1410278032	10	0.03	0	13	0.2	0.04	5	No
1410278033	4.6	0084	1	13	0.2	0.04	5	No
1410278034	5	0.07	2	13	0.2	0.04	5	No
1410278035	17	0.04	3	13	0.2	0.04	5	No
1410278036	39	0.26	0	39	0.2	0.04	5	Yes
1410278037	18	0.21	0	18	0.2	0.04	5	Yes
1410278038	5	0.13	1	18	0.2	0.04	5	No
1410278039	13	0.05	2	18	0.2	0.04	5	No
1410278040	13	0.05	3	18	0.2	0.04	5	No
1410278041	1	0.17	4	1	0.2	0.04	5	Yes
Avg	12.28	4.31	-	14.59	-	-	-	30%

As we can see from table 3.1, the CPU utilization has obviously changes at time stamp 1410278026, 1410278027, 1410278036 and 1410278037, respectively. These changes are greater than the set value of threshold. The condition 1 of monitoring strategy is met and the key change flag is marked at these time stamps. The freshness of the data on monitored node is synchronized to monitoring component. Meanwhile, at time stamp 1410278031 and 1410278041, the changes are greater than the set value of the minimum threshold during the interval time. It meets the condition 2 of monitoring strategy and is looked as the key change as well.

Other changes are considered non-key changes, and cannot trigger the data transmission, monitoring data remains the same and without any update. Monitor node and monitoring components CPU utilization curves are shown in Figure 3.4.

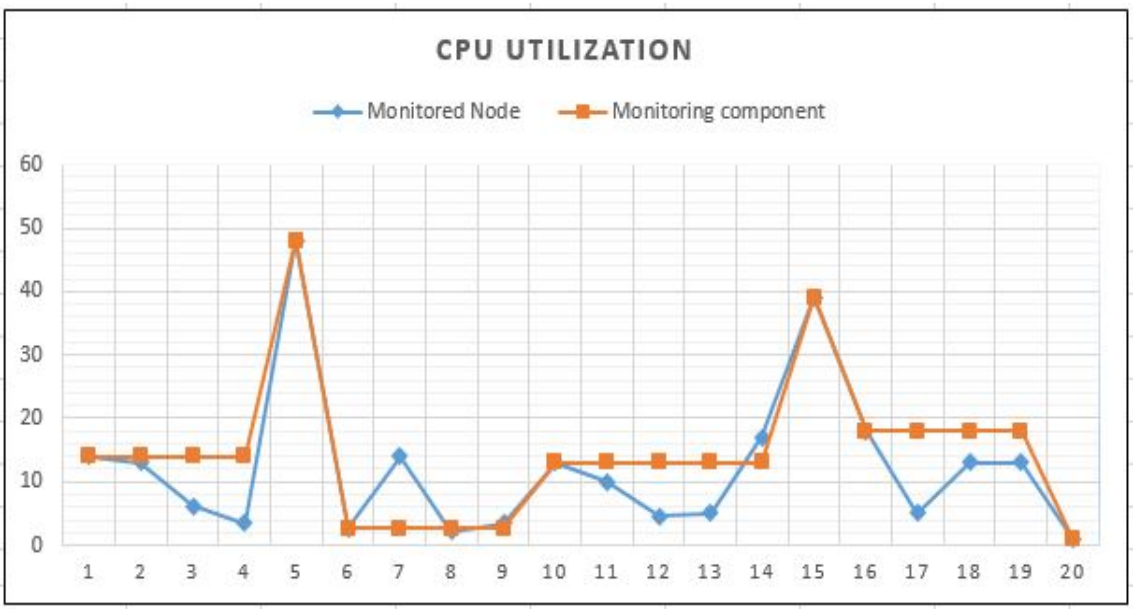

Figure 3.4: CPU Utilization between monitored node and monitoring component

3.2.2 Threshold and Interval

Exponentially Weighted Moving Average (EWMA) is a common method to process data sequences. It calculates values to get exponential smoothing prediction model based on historical data. Since historical data on different time stamps have certain influence on the current forecast points; therefore, the weighted influence of each value shows an exponential decrease with time.

We use EWMA to predict the load change of the monitored resource and the time interval between two adjacent key changes. Since the value of threshold and interval reflect the trends of monitoring data sequence, it can be used to predict and identify key changes. The selection of appropriate value not only has the effect of modifying the prediction results, but also can effectively predict the trend of the dynamic changes of resource load.

The advantage of dynamic adjustment parameters is to adapt to the real situation of resource load, achieve dynamic adjustment for the frequency of data transmission and to ensure efficient communication between monitored resources and monitoring components. The equations are shown in (3.3) and (3.4) [7]

$$threshold\_E_t = \alpha \cdot threshold_{t-1} + (1 - \alpha) \cdot threshold\_E_{t-1} \tag{3.3}$$

At any time t, the $threshold\_E_t$ represents the predictive value of load change on the monitored resource, $threshold_{t-1}$ refers to the actual observed value for the closest time stamp prior to time t, α is defined as the weight of the threshold value between the last observed value and predicted value. The range of α is [0,1]

$$interval\_E_t = \alpha \cdot interval_{t-1} + (1 - \alpha) \cdot interval\_E_{t-1} \tag{3.4}$$

$interval\_E_t$ is the predicted value of the time interval of load change, $interval_{t-1}$ is the actual observed value of the time interval for the closest time stamp prior to time t.

The validity of prediction depends on the value of α, which describes the stability of the monitored resources state. A small α indicates that the load changes smoothly, the state of resource is stable. On the opposite side, a large α indicates that the load changes quickly. Therefore, according to the load changes select the appropriate value of α.

3.3 Push-Pull Hybrid Model (PPHM) Design

The PPHM model consists of push and pull algorithm. It is implemented at monitored nodes and monitoring components, respectively. In order to keep the data consistency and reduce the data transfers between the monitored nodes and monitoring components, the load change of resources are divided into key and non-key changes at monitored nodes. Only the key changes can be transferred to ensure timely

and accurate resources information in the server side and non-key changes are ignored to reduce the update times.

At monitoring sides, EWMA method is used to predict the trend of resource load, identify key changes and adjust transmission time interval to reduce communication overheads. The EWMA prediction of results are used to pull model for the time interval of load change. The push-pull hybrid model is composed of push algorithm and pull algorithm.

3.3.1 Push Algorithm

Push algorithm is the main algorithm for resource monitoring that is used to predict the trends of resource on monitored node, and proactively identify and transfer the key changes.

The main idea of push algorithm is described in Figure 3.5.

- If the load change is greater than zero on the monitored resource, the equation 3.4 is called again to calculate the time interval and predict resource load change for the next time interval.
- If the load change is greater than threshold that is considered as the key change, then resource load information and interval will be transferred regardless of the timer expires or not. The equation 3.3 is called to calculate the threshold, predict the next key change and restart the timing.
- Otherwise, ignore the update of resource load information and reduce data transmission. If the timer expires, then restart the timer. Meanwhile, at the same time, check whether the load change is greater than $min\_threshold$ or not. If so, it is considered as the key change and actively transmit the load information and interval, recalculate the threshold and restart the timing.

Assuming at a certain time, the resource load is $P(t_p)$ at monitored node, $C(t_c)$ is the resource load information that is saved at the monitoring component, the load change amount is the absolute value of the differential between $P(t_p)$ and $C(t_c)$.

In most situations, push operation runs before the pull operation. Hence, the push-based method is dominant.

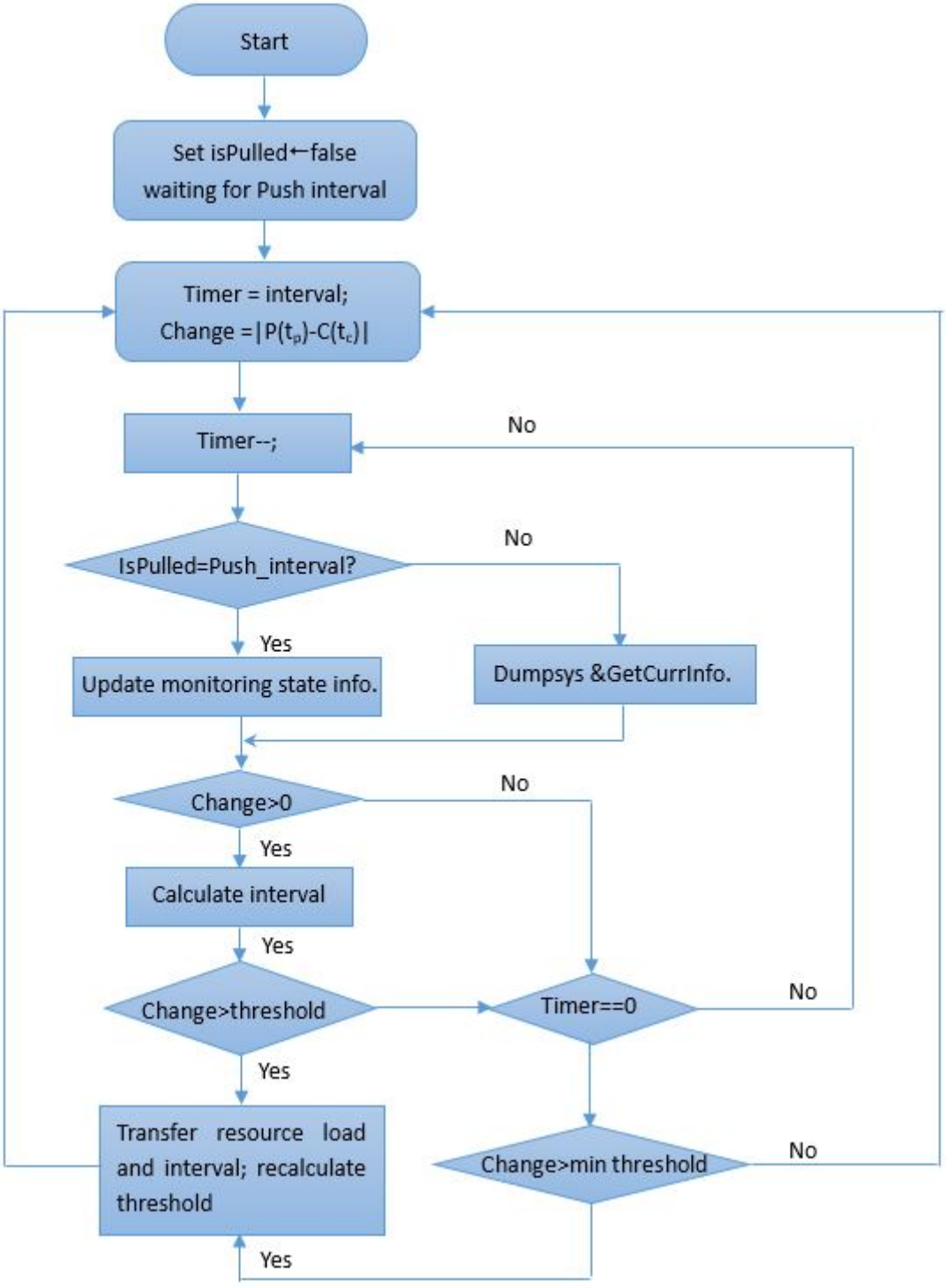

Figure 3.5: Push Algorithm Flow Chart

3.3.2 Pull Algorithm

The push algorithm runs at the monitored nodes and the pull algorithm runs at the monitoring nodes simultaneously. The two algorithms try to make the resource monitoring system intelligently switch between push and pull operations according to user's requirements and status' changes of monitored resources.

The main idea of the pull algorithm is that the monitoring component that receives resource load information, also stores the interval time from the push algorithm. The interval is kept for the next time when the monitoring component receives information, if the update loading information is not received during the time interval by the monitoring component. The monitored node is considered as disable or unstable state. Then the monitoring component timely sends a query to the monitored node for the latest status.

The pull operation identifier "isPulled" and push identifier "isPushed" are set to be mutually exclusive to avoid push and pull operations concurrently happening at the same time. For example, if pull operation happens, push operation is forbidden in the corresponding time interval, and vice versa. This is to say, when the value of threshold is equal to 0, the model has the highest consistency (real-time synchronization); all of pull operations are forbidden, and the hybrid model degrades to the pure push model.

Similarly, when the threshold of value is equal to 1, the model has the worst consistency; all of push operations are forbidden, and hybrid model degrades to pure pull model. Also, "isPulled" and "isPushed" should be controlled by synchronization model to avoid inconsistency during push and pull operations.

When the value of threshold is relatively small, the push method dominates. According to Figure 3.5, the push algorithm is easily to be met, push operations are frequently triggered. On the other side, although the pull algorithm is trying to minimize Pull interval's value, the PULL_INTERVAL_MIN blocks this trend when pull interval becomes very small.

In most situations, push operation runs before pull operation. Hence, the push-based method becomes dominant. If push operation is not triggered for a long time, the pull algorithm sets the pull time interval as PULL_INTERVAL_MAX to inform availability of the monitored to the monitoring nodes.

The pull algorithm is described as following.

```
# Procedure Pull Model
while (true)
do
        Set isPushed <==Flase
        waiting for Pull_interval
        if isPusheded = True
                Push load info. to monitoring component
        else
                isPulled <==True, send Pull query
                update Load info. to monitoring component
        endif
Set getLoaddiff() = change
if (change <= threshold)
        if increased_Pull_interval <= Prediction
                Pull_interval = increased_Pull_interval
        else
                Keep current Pull_interval
        endif
else if (change > threshold)
                if decrease_Pull_interval > Prediction
                        Pull_interval = decreased_interval
                else
                        Keep current Pull_interval
                endif
        endif
update Load info on Monitoring Component
Send Current Load info. to Monitoring Component
endwhile
```

3.4 Resource Information Definition

Monitored state information can be classified into three categories by the resource load. Each state is defined as follows:

- S(Stable State) stands for an operation available state.
- U(Unstable State) stands for an operation available but discontinued state.
- D(Disable State) stands for an operation unavailable state due to faults or network disconnection.

The three state values are set based on CPU utilization (C_{util}), this classification comes from the CPU Utilization Guide of IBM [19]. CPU utilization is the most important metric, since the system performance will dramatically drop if application processing saturates the CPU when CPU utilization is over 90%.

$$\begin{aligned} S_{st}: &\quad 0\% \leq \ C_{util} \ \leq 70\% \\ U_{st}: &\quad 70\% \leq \ C_{util} \ \leq 90\% \\ D_{st}: &\quad 90\% \leq \ C_{util} \end{aligned}$$

The information is collected from mobile devices that is used to calculate the utilization rate of each mobile cloud resource. The utilization of CPU is calculated by the following formula. The utilization of other resources can be calculated in the same way, such as memory, storage, and battery. It is written with M_{util} and B_{util}.

$$C_{util} = \frac{C_{user}+C_{sys}}{C_{total}} \times 100$$

$$M_{util} = \frac{M_{user}+M_{cache}}{M_{total}} \times 100$$

$$B_{util} = \frac{B_{cur}}{B_{total}} \times 100$$

It should be noted that the memory management on mobile device is quite complicated, and the concepts behind them are quite complicated too. To calculate how much memory is used by the system, Android uses the Proportionate Set Size (PSS), a statistic that the Android system computes to determine if it is going to kill the process. Basically, it is a sum of the non-shared memory consumed by system and App.[1]

The state information of mobile resources is changed according to utilization of system (CPU, memory, battery, storage usage, and bandwidth) and location. The *EWMA* algorithm is used to predict the next state and calculates a monitoring time interval according to the predicted states.

[1]Memory information will be further analysed at Chapter 4.

Chapter 4

Results and Discussions

To verify the performance of the different data transmission models, an Android based experiment environment is built to implement a dynamic resource listening system. The system is capable of identifying and observing the key change degree for the load of monitored resources on the phone. The monitoring records are composed of name of the monitoring source, usage of the resource and a timestamp of when the resource was dumped by Android Debug Bridge *(ADB)*. Meanwhile, a monitoring record message is sent to the monitoring side for further state prediction. The system can intelligently switch between push and pull operations according to the status changes of the monitored resources.

Relative to the original problem statement: the primary motivation behind the implementation of the monitoring system is to observe, analyze and make full use of the system resource utilization on mobile devices in the mobile cloud environment. The typical data transmission model (such as push mode, pul1 mode, and push-pull hybrid model) are respectively applied to the monitoring system for comparative analysis. The updating rate of the models is considered as main evaluation indicator to compare and analyze the experimental results. Potential usage of the interesting findings will also be discussed. This section will start with experiment setup followed by analysis of results.

4.1 Experiment Setup

The overall resource monitoring for Android mobile system is implemented in three *(Samsung Galaxy Ace II)* phones with *v4.1.2 "Jelly Bean"* and a *Samsung Galaxy*

Tab10.1 P7510 on the Android *v4.0 "Ice Cream Sandwich"*. The purposes behind picking these devices is to avoid any inconsistency and compatibility issues that are caused by differential between the hardware's performance and *OS* version. We used an HP Pavilion G6 as a data center server to receive monitoring data, trigger models, and evaluate the performance of the *push&pull Hybrid Model (PPHM)*. The Notebook is equipped with AMD CPU A6-3400M@4*1.4GHz, 8GB RAM and Windows 7 64-bit operating system. The communication between each Android device and the server works independently. This is to say, the monitored nodes and the server only communicate with its partner within the pair.

Android Debug Bridge (ADB) is adopted to dump system information and to perform resources for each device. Terminal IDE is used as a platform to write *ADB* command lines directly on the Android device. A rooted *BusyBox* is used to set up the scheduling tasks because there is no standard Java schedule like the *TimerTasks* class on android devices. The scheduling jobs are implemented by Java programming language with Java synchronization and multi-thread mechanisms. To simplify the experiment, we only output some of the resource parameters, i.e. the CPU, memory load percentage and battery usage, to test performance of the P&P model.

After the tools are installed on each device, two different data transmission process are performed where the pull model is to implement active data collection, and the push is to implement passive data collection model. The idea is to observe and compare the differences of the two models, and verify the hybrid model to keep high data consistency and reduce communication overhead. So our experiments will analyze and evaluate the P&P model with respect to the above two aspects.

4.2 Results

In order to evaluate the performance of data transmission models, this paper adopts the number of updates and consistency as the evaluation metrics. In order to verify the performance of P&P Hybrid model, the monitoring modules adopt a push mode, pull mode, and PPHM model for data transmission respectively, and compare monitoring performance that is based on the data update rate and consistency for different data transmission models.

4.2.1 Resource Utilization

Figure 4.1, is a display of the CPU and memory utilization that is collected by the mobile device using the monitoring models from 1410278081 to 1410814167. The x-axis shows the number of each monitoring time stamp. The y-axis marks the percentage of CPU and memory utilization.

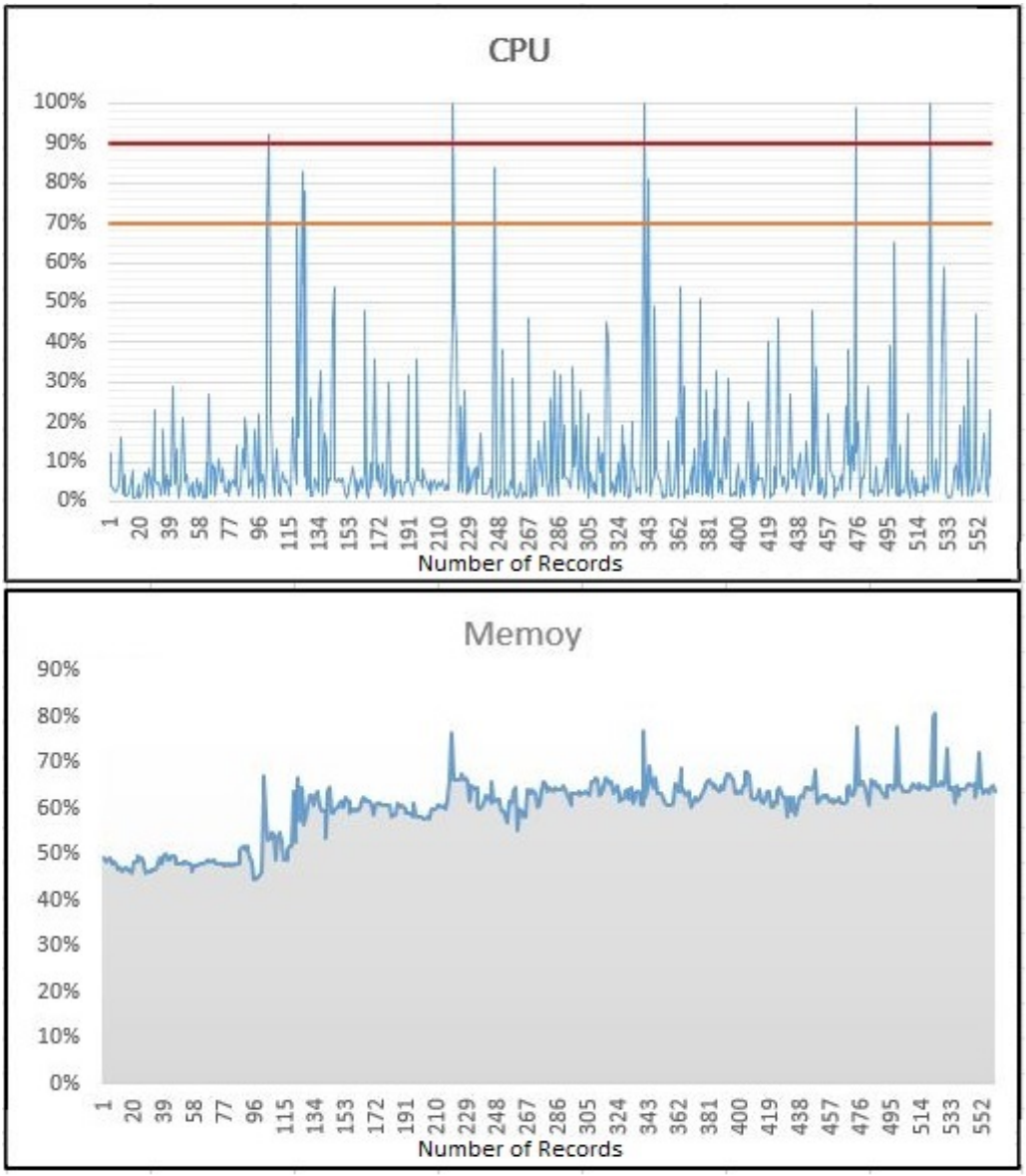

Figure 4.1: CPU&Memory Information in Mobile Device

As it can be observed from the figure, most of the time (97.8%) mobile phone resources are in idle state when the utilization rate of resources is between 0% and

70%. Around 1.1% mobile phone resources are in an unstable state when the utilization rate of resources is between 70% and 90%. Another 1.1% resources are in the disable state when the utilization rate of resources is up to 90%. One interesting observation in the experiment is that the utilization between CPU and memory are highly consistent, and the state also shows a high consistency between them.

Compared with CPU, Android system has a higher memory utilization because after starting the *init* process, it will load the *runtime* and *Zygote* process, and *SystemServer* will also start all the system core services. About 98 percent of the time, the memory is in a stable state and the usage is between 50% and 70%. Once the mobile device is considered as a provisioning cloud resource, the utilization of memory will become a bottleneck affecting system performance soon.

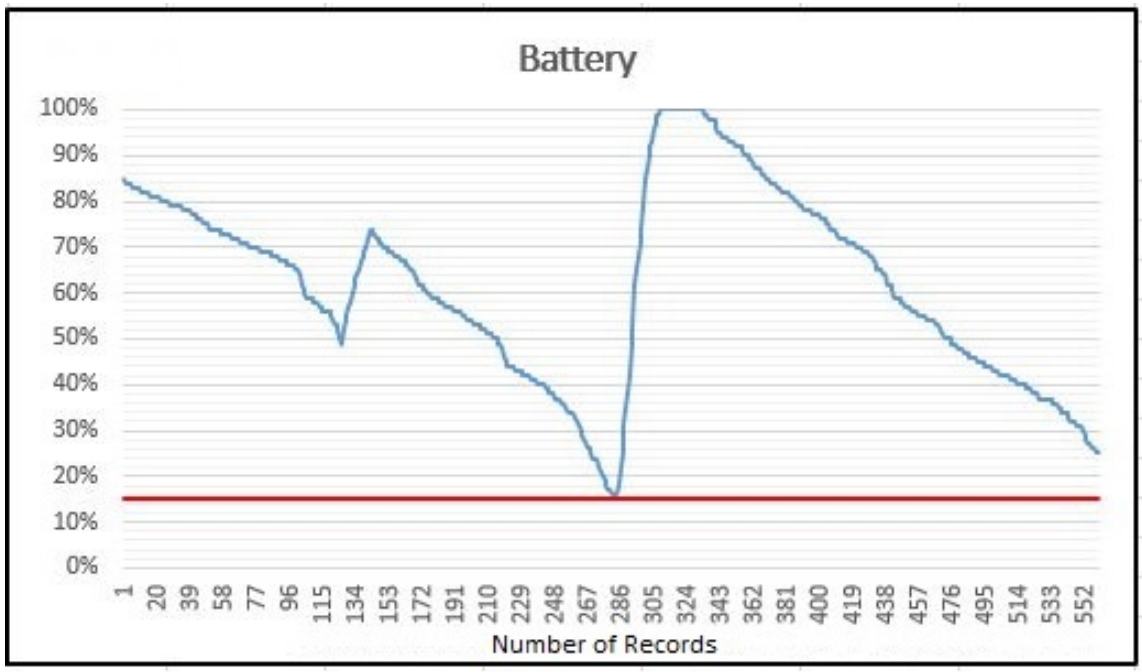

Figure 4.2: Battery Information in Mobile Device

Figure 4.2 shows the percentage of battery resource change with different time stamps. The x-axis displays the number of each monitoring time stamp. The y-axis displays the rate of battery utilization. It could be observed that the discharge and recharging process of battery is not linear. In the experiment, we set up an automatic alarm mechanism where a notification will be sent to user once the power is less than 15 percent. The purpose is to avoid the undergoing process be unpredictably interrupted since the limitation of power supply.

4.2.2 Models Analysis and Evaluation

The following experiments' purpose is to reveal the relationship between the updating number and threshold, and compare updating number of the P&P model with the pure push and pure pull models. We run *dumpsys* tool periodically on 180s to retrieve system information. It begins at 1410278081 and ends at 1410814167 with 560 times of updating. The pull interval is set to 180s, and the PULL_INIT_INTERVAL, PULL_INTERVAL_MIN, and PULL_INTERVAL_MAX are set to 50s, 30s and 120s, respectively. The pull interval increases or decreases 10s each time.

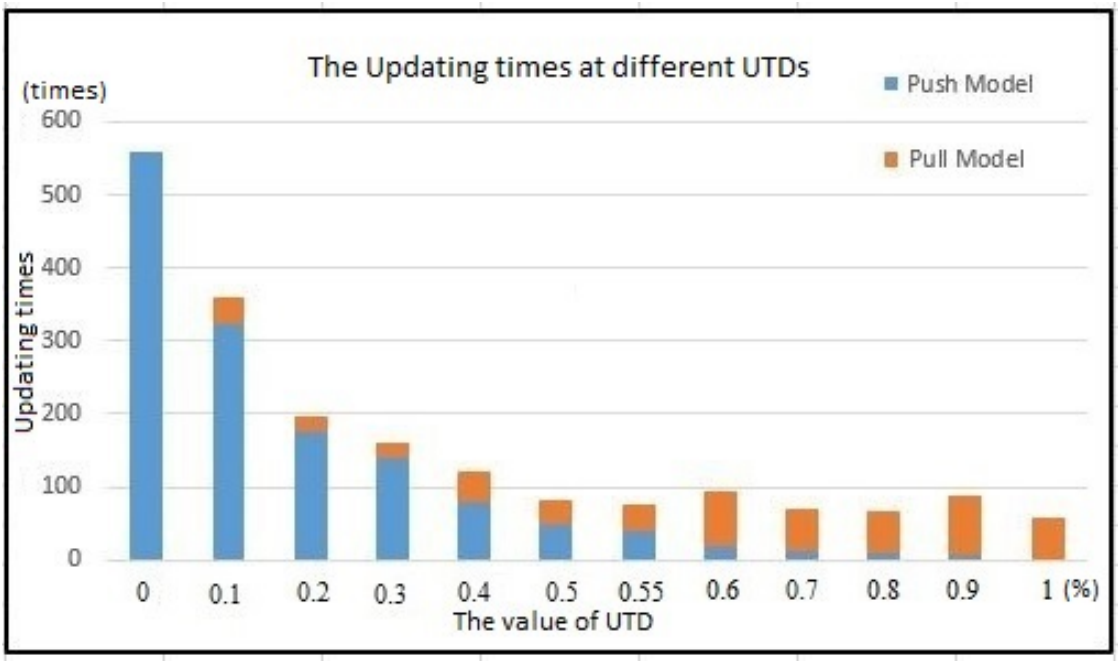

Figure 4.3: Updating Number of P&P Model at different UTD value

The x-axis displays the threshold that is defined as User Tolerant Degree (UTD). A small UTD represents that the accuracy requirement is paid more attention rather than communication overhead on P&P model. On the opposite, a large UTD indicates that a significant level of inaccuracy can be accepted during the data transmission. The y-axis displays the CPU load. The color scheme corresponds to the categories of data transmission.

The total updating number includes both the numbers of push and pull operations. As we can see from Figure 4.3, the total updating number is heavily determined by the value of UTD. When the value of UTD rises, the total updating number decreases, the number of push event becomes to dramatically drop. Meanwhile, as a supplementary

method, the pull event starts to dynamically check the living system information from monitored mobile devices, so the number of pull operation grows slightly. The experiment result is a good match to our expectation that the total number of pull operations increase is much less than the number of push operations decrease. After all, push method dominates the operations for data collection. Comparing both the push and pull operations in Figures 4.3 and 4.4, the proportion of push operation decreases, while pull operation accounts for more of the total as the value of UTD grows.

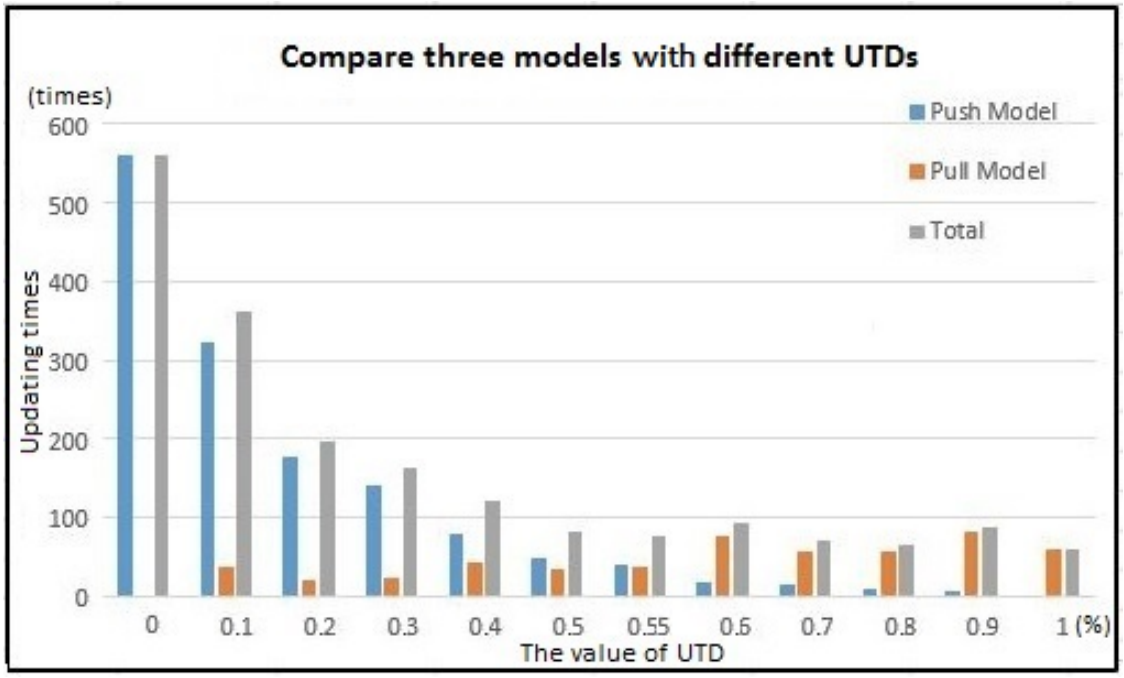

Figure 4.4: Updating times of three models in different UTDs

In Figure 4.3, when the value of UTD is set to 0, the P&P model degenerates to the pure push model where the pull operation is forbidden, the communication overhead reaches to the highest level. When threshold is relatively low, for example, the value of UTD is set to 0.1 or 0.2, the push event is dominant, most of the operations are push, whereas pull operations only occupy a little percentage. The communication overhead is still high but high accuracy and consistency is kept between monitored devices and the server.

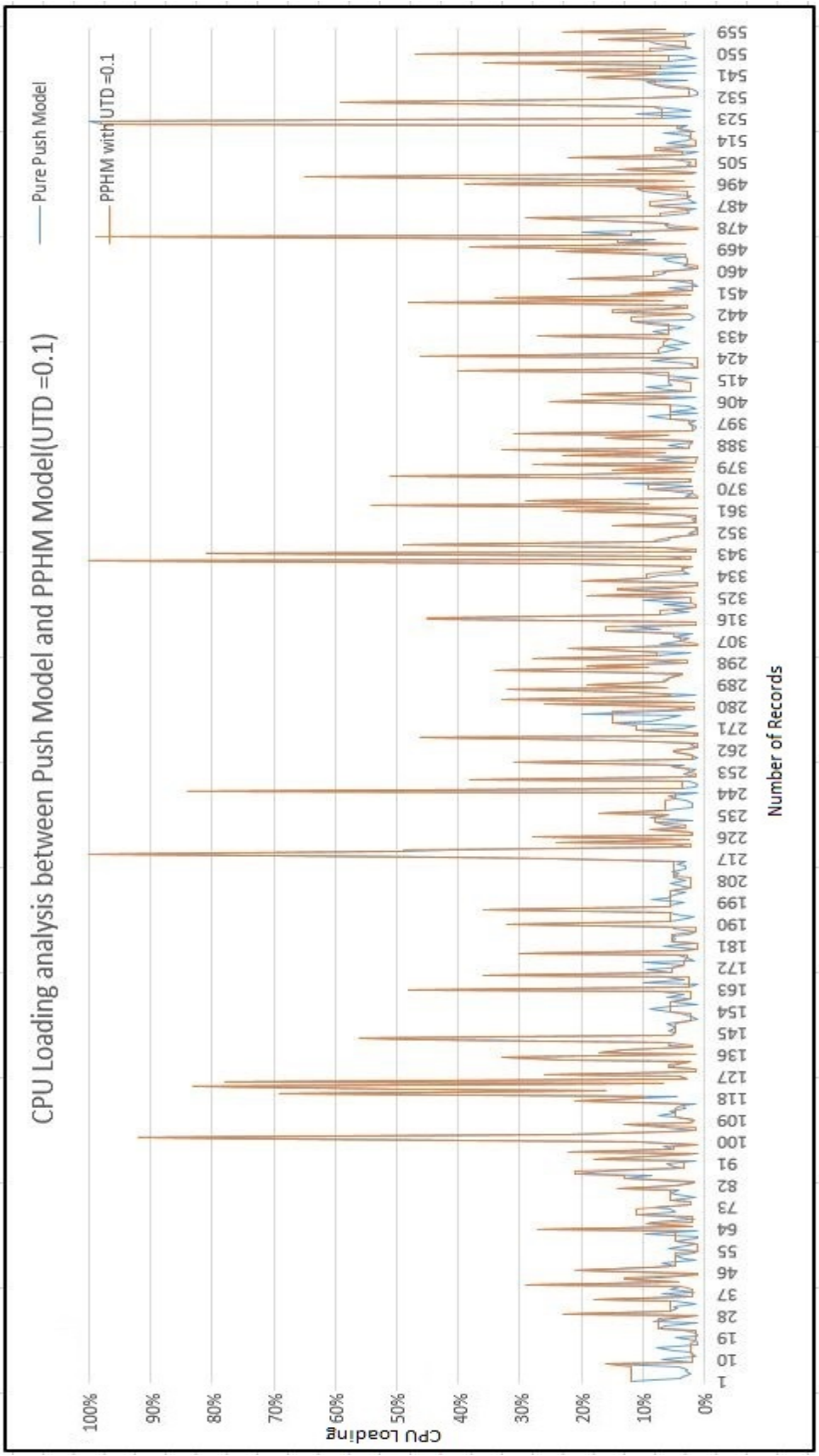

Figure 4.5: Compared CPU Loading between Push, Pull Model and PPHM Model (UTD = 0.1)

If the value of UTD is relatively moderate, none of the push or pull is dominates. For example, UTD is set to 0.5, 0.55 or 0.6, the total updating number drops sharply. This scenario is just in the middle of the two models, and both push and pull methods act frequently. The communication cost achieves the minimum value, but the tradeoff is that the monitoring data is lagging behind the server side. When the value of UTD(0.7-1.0) is relatively large, in this scenario, the pull-based method becomes dominant. push operation is seldom triggered. Overall, the number of pull operations greatly outweighs that of push operations when UTD is set greater than 0.6. When the value of UTD is set to 1, the PPHM model becomes the pure pull model.

In Figure 4.5, we compare the updating number between the P&P model and pure push model. The x-axis is the number of resource monitoring time stamp. The y-axis displays the percentage of CPU utilization. To observe the correlation between the PPHM and the pure push model, the updating number of PPHM model is much smaller than the updating number in the pure push method. It is obvious that P&P model accurately captures all of the key changes and successfully ignores non-key changes during the experiment. As we can see, the PPHM model can save 36.4% communication cost by introducing 0.1 UTD. The push operations on PPHM are decreased to 43.4% compared to the pure push model. The pull operations only increases 6.7%. The reason why the total updating number of transmission significantly reduced is that the trivial, non-key changes are ignored and only transfer the predefined threshold of the key changes. In the peak sections, the line of PPHM can be fully fitted with push operations; at the bottom sections, the fluctuation of line is represented by the trend of the pull operations. In summary, the PPHM model performs strong coherence with the pure push method when UTD is relatively low (0.1), also high accuracy and low communication cost are achieved by reducing unnecessary push operations.

Comparing the pure push and pull operations, the PPHM model can intelligently switch the two models according to users' consistency and accuracy requirements. The detailed operations of push, pull and P&P model is displayed in Figure 4.6. The y-axis is the percentage of CPU utilization, and the x-axis is the time. The square points represent push operations, and the triangle points represent pull operations. Although the graph loses more minor details, it is still much similar to the pure push model with less number of updating times.

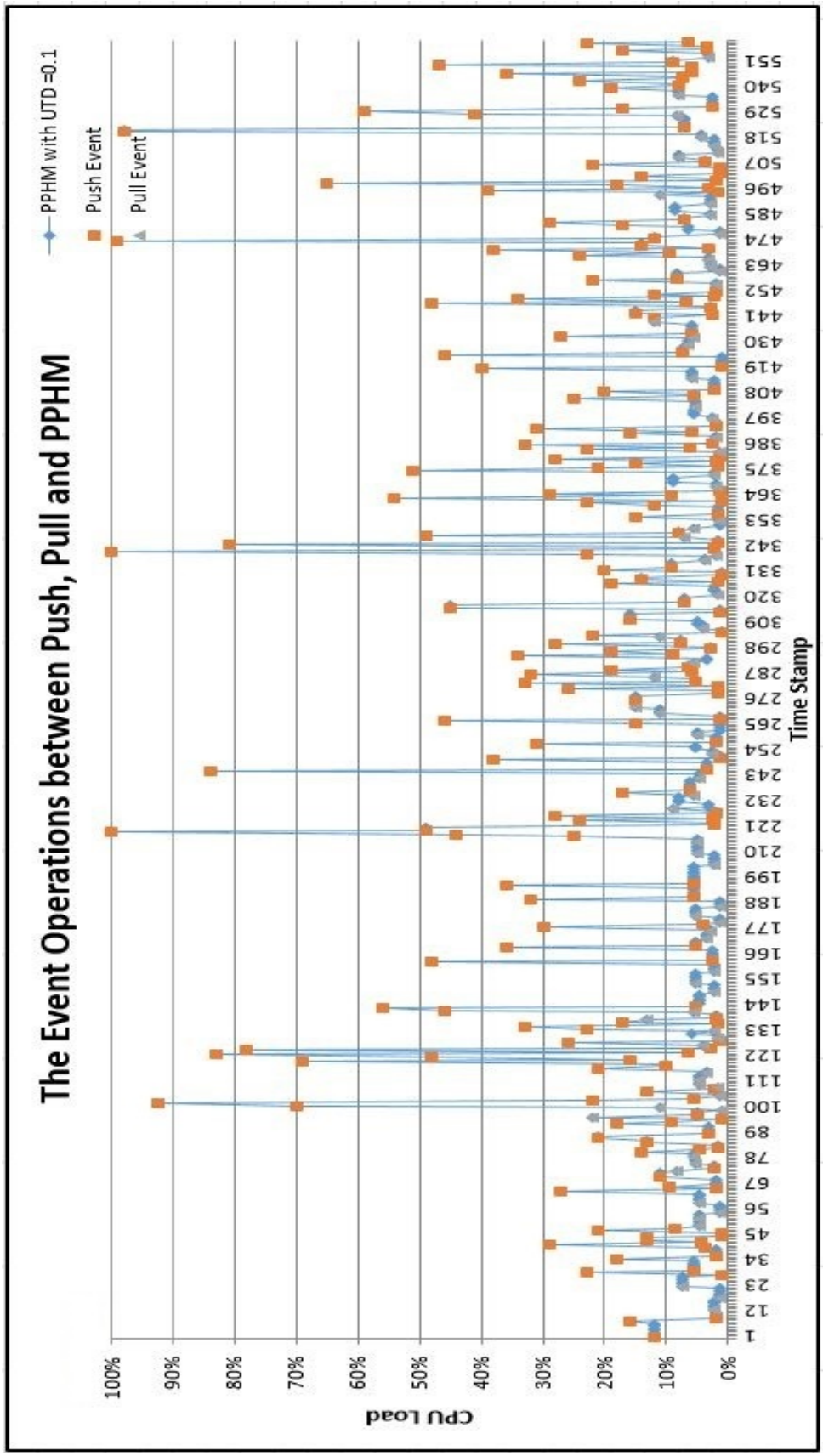

Figure 4.6: Detail Operations Event between Pure, Push Model and PPHM Model (UTD = 0.1)

From the global view, the push method dominates the monitoring strategy in PPHM model. This is because push operations are usually triggered at points that have remarkable status changes. This is to say, push model can accurately identify sudden key changes regardless of at the peak or at the bottom. In contrast, the main purpose of pull method is to detect living signals from mobile devices, so pull operations happen somewhat randomly.

It is very clear from Figure 4.4 and figure 4.6 that the turning point for push and pull operations occurs when UTD is set to 0.55. After this point, the number of pull operations greatly outweighs that of push operations when the value of UTD is between 0.6 and 1. Because the user may only want to know the approximate load information about the system resources, it involves a small number of push operations. It deviated from the consistency and accuracy of the requirements, so I did not do further discussion in here.

Overall, the PPHM model performance has strong coherence to the pure push method when UTD is relatively low, around the half number of push updating is dropped by introducing 0.1 UTD. Although the updating of number and communication costs are significantly decreased as the UTD increases (0.6-1), some noticeable status changes will be missed due to the coarse resources loading requirement.

Chapter 5

Conclusions and Future Work

This chapter summarizes the project by outlining the contributions and presenting ideas for future improvement of the push and pull hybrid model *(PPHM)*.

5.1 Summary

The project proposes a useful push and pull hybrid model which inherits the advantages of push and pull models for monitoring mobile devices. The first chapter of this report introduced the concept of cloud computing, monitoring system architecture, service model, and then focused on the data transmission model of monitoring technology to adapt heterogeneous and dynamic characteristics on mobile devices. We compared and analyzed some of the mobile Cloud monitoring methods in Chapter 2. After analyzing the architecture of Android, we adopted *ADB Shell Commands* to capture the information about the status of system services. In Chapter 3, we discussed the benefit of data transmission mode for push and pull. Based on related literature ideas and methods of data transmission, this report proposed a push and pull hybrid model *(PPHM)* that combines the advantages of push mode and pull mode. Chapter 3 presented a step-by-step design and implementation for the push-pull hybrid model. Firstly, we defined the concept of Key and Non-key changes and setup condition to determine the Key change. The resources load was classified as Key and Non-key changes. Secondly, we introduced Exponentially Weighted Moving Average *(EWMA)* method to predict the trends of monitored resources according to the historical information. Finally, the algorithm of push-pull hybrid model was presented to adjust the transmission time interval and reduce communication overheads.

In the push algorithm, the monitored nodes can actively recognize and push the key changes that can impact the system consistency at different transmission intervals. The frequency of data transmission can adapt to the resource load change tendency, keep the monitoring component and the monitored nodes continuous communication, ensure the monitoring information real-time synchronization. In the pull algorithm, when the monitoring component does not receive loading resources information within the predetermined time interval, then the pull mode can actively query the loading resource to keep abreast of the performance of the monitored node, and early detect the faults. In order to prove the feasibility of the PPHM model, the testbed was built to evaluate the monitoring system in Chapter 4. The push, pull and PPHM algorithm was applied to the monitoring module, and the monitoring performance is based on the update rate and consistency. The comparison results showed that the push-pull hybrid model can improve the data consistency and reduce communication overhead, and its performance was better than the traditional data transmission model. In Chapter 5, we evaluated the relation between updating number and UTD, and compared the updating number of the PPHM model with the pure push and pull models. We concluded that the PPHM model can effectively reduce updating number and maintain various levels of coherence in accordance with the users requirements..

5.2 Future work

As for future work there are many things that could be done in the following areas. For the monitored side, fine grain filters could be implemented to decrease the size of data collection during the push process. It can also help reduce data transmission communication overhead between push and pull model. For the server side, a graphical web interface could be created to display the data collected.

The experimental results show that the model can keep data consistency, reduces the communication overhead, and improve the efficiency of monitoring performance. But it only reflects the resource's usage at a specific timestamp, cannot send an alarm when the mobile device has performance bottleneck or an early failure occurred, and therefore PPHM model has a bit lag behind compared with pure push model. Further, in practical mobile cloud platforms, which consists of the large number of mobile devices, it will generate huge amounts of monitoring data as time goes on. It is a big challenge to read, write, store and query big data. Lastly, for analysis, the experiment could be tested on multiple devices to observe cross device behavior.

Finally, data mining could be used on monitoring result to identify Key or Non-Key change. Overall, this is a small step in contributing datasets and presenting creative ways so that the datasets could be used to solve real world problems.

Appendix A

Glossary

Mobile Cloud Computing (MCC) is the combination of cloud computing, mobile computing and wireless networks to bring rich computational resources to mobile users, network operators, as well as cloud computing providers.

Resource Monitor is to collect and display the usage of hardware (CPU, memory, disk, and network) and software resources in real time.

Rooting is the process of allowing users of smartphones running the Android mobile operating system with superuser permissions.

HTTP POST Request method is designed to request that a web server accepts the data enclosed in the request message's body for storage.

Android Debug Bridge (adb) is a versatile command-line tool that lets you communicate with a device (an emulator or a connected Android device).

Dumpsys runs on the device and provides information about the status of system services.

User Tolerant Degree (UTD) describes how tolerant a user for the status inaccuracy.

The change degree describes the extent of change between the current status of a monitored resource and the status preserved in the corresponding monitoring nodes.

Bibliography

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy H Katz, Andrew Konwinski, Gunho Lee, David A Patterson, Ariel Rabkin, Ion Stoica, et al. Above the clouds: A berkeley view of cloud computing. 2009.

[2] Kapil Bakshi. Cisco cloud computing-data center strategy, architecture, and solutions. *DOI= http://www. cisco. com/web/strategy/docs/gov/CiscoCloudComputing_WP. pdf*, 2009.

[3] Stefan Brahler. Analysis of the android architecture. *Karlsruhe institute for technology*, 7:8, 2010.

[4] Developed by GXPMEDIA.COM. MonitorTools.com cloud monitoringing software, 2015.

[5] Jason Carolan, Steve Gaede, James Baty, Glenn Brunette, Art Licht, Jim Remmell, Lew Tucker, and Joel Weise. Introduction to cloud computing architecture. *White Paper, 1st edn. Sun Micro Systems Inc*, 2009.

[6] 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, 2014.

[7] Peter A Dinda. The statistical properties of host load. *Scientific Programming*, 7(3-4):211–229, 1999.

[8] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[9] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *2008 Grid Computing Environments Workshop*, pages 1–10. Ieee, 2008.

[10] Dijiang Huang, Tianyi Xing, and Huijun Wu. Mobile cloud computing service models: a user-centric approach. *IEEE Network*, 27(5):6–11, 2013.

[11] He Huang and Liqiang Wang. P&p: a combined push-pull model for resource monitoring in cloud computing environment. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 260–267. IEEE, 2010.

[12] Markus C Huebscher and Julie A McCann. A survey of autonomic computingdegrees, models, and applications. *ACM Computing Surveys (CSUR)*, 40(3):7, 2008.

[13] Angela Song-Ie Noh, Eui-Nam Huh, Ji-Yeun Sung, and Pill-Woo Lee. An optimal and dynamic monitoring interval for grid resource information system. In *Proceedings of the 2005 International Conference on Computational Science and Its Applications - Volume Part I*, ICCSA'05, pages 1144–1153, Berlin, Heidelberg, 2005. Springer-Verlag.

[14] JiSu Park, KwangSik Chung, EunYoung Lee, YoungSik Jeong, and HeonChang Yu. Monitoring service using markov chain model in mobile grid environment. In *International Conference on Grid and Pervasive Computing*, pages 193–203. Springer, 2010.

[15] JiSu Park, HeonChang Yu, KwangSik Chung, and EunYoung Lee. Markov chain based monitoring service for fault tolerance in mobile cloud computing. In *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, pages 520–525. IEEE, 2011.

[16] Ju Ren, Yaoxue Zhang, Kuan Zhang, and Xuemin Shen. Exploiting mobile crowdsourcing for pervasive cloud services: challenges and solutions. *IEEE Communications Magazine*, 53(3):98–105, 2015.

[17] Aubrey-Derrick Schmidt, Rainer Bye, Hans-Gunther Schmidt, Kamer Ali Yüksel, Osman Kiraz, Jan Clausen, Karsten Raddatz, Ahmet Camtepe, and Sahin Albayrak. Monitoring android for collaborative anomaly detection: A first archi-

tectural draft. *Technische Universität Berlin-DAI-Labor, Tech. Rep. TUB-DAI*, 8:08–02, 2008.

[18] WA Vogels. Head in the cloudsthe power of infrastructure as a service. In *First workshop on Cloud Computing and in Applications (CCA08)(October 2008)*, volume 5, 2008.

[19] James A Whittaker and Michael G Thomason. A markov chain model for statistical software testing. *IEEE Transactions on Software engineering*, 20(10):812–824, 2004.

[20] Chanmin Yoon, Dongwon Kim, Wonwoo Jung, Chulkoo Kang, and Hojung Cha. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pages 387–400, 2012.

[21] Serafeim Zanikolas and Rizos Sakellariou. A taxonomy of grid monitoring systems. *Future Generation Computer Systems*, 21(1):163–188, 2005.

[22] Leah Zhao, Neil Wong Hon Chan, Shanchieh Jay Yang, and Roy W Melton. Privacy sensitive resource access monitoring for android systems. In *2015 24th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6. IEEE, 2015.