Benefits of Utilizing an Edge Server (Cloudlet) in the MOCHA Architecture

by

Zuochao Dou

Submitted in Partial Fulfillment

of the

Requirements for the Degree

Master of Science

Supervised by

Wendi B. Heinzelman

Department of Electrical and Computer Engineering
Arts, Sciences and Engineering
Edmund A. Hajim School of Engineering and Applied Sciences

University of Rochester
Rochester, New York

2013

Curriculum Vitae

The author was born in Beijing, P. R. China in 1986. He joined Beijing University of Technology in 2005, where he received a B.S. degree in Electronics in 2009. From 2009 to 2011, he studied at University of Southern Denmark, concentrating on embedded control systems. Then, he began his graduate studies at the Department of Electrical and Computer Engineering at the University of Rochester in 2011. He is currently working towards his M.S. degree in the area of mobile-cloud computing with the guidance of Professor Wendi B. Heinzelman.

Acknowledgments

I would like to thank Professor Wendi B. Heinzelman, my research and thesis supervisor throughout my masters studies. Her erudition, vision, as well as the rigorous attitude in research impress and inspire me all along. In addition, I am thankful to Professor Minseok Kwon and Professor Tolga Soyata for their suggestions and consistent support during this thesis.

I would also like to thank all of my colleagues in the MOCHA research group. You have helped me overcome barriers in my research and provided an excellent and supportive environment to work in.

Finally, I would like to thank my parents, my friends and my girlfriend. You have provided me with an overwhelming amount of support for which I am forever grateful.

This research was funded in part by UCB Pharma and by CEIS, an Empire State Development-designated Center for Advanced Technology.

Abstract

In mobile-cloud computing, there are two primary hurdles: (1) long communication latency and non-negligible communication latency uncertainties from a mobile to the cloud; and (2) optimal server(s) selection from among the large number of available cloud servers. The mobile-cloudlet-cloud architecture called MOCHA (MObile Cloud Hybrid Architecture) introduces an intermediary (cloudlet) between the mobile devices and the cloud servers. The introduction of a cloudlet, which serves as an edge-server, provides us the potential to address these hurdles.

The goal of this thesis is to demonstrate the benefits (in terms of communication latency) in utilizing the edge-server (cloudlet) in the MOCHA architecture for mobile-cloud computing through the architecture hardware and software setup of MOCHA, network latency measurements, data analysis and modeling, as well as task division algorithm simulations and validations.

The results show that utilizing a cloudlet in mobile-cloud computing provides significant benefits from the use of dynamic profiling (utilizing two algorithms called fixed and greedy) for optimal server(s) selection. On the other hand, without profiling, a random server selection approach is capable of providing acceptable latency performance with high redundancy.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Motivation

In the past decade there have been incredible advancements in mobile, wireless, and cloud technologies. Cloud servers are available everywhere, allowing users to remotely access enormous computing power and storage. These resources can be utilized to handle computationally intensive applications that can hardly be run on mobile devices with limited resources, such as tablets and smartphones. Thus, it is desirable to offload the computation to cloud servers. Long latency and huge latency uncertainties from a mobile to the cloud are often referred to as a dominating technical hurdle for the widespread adoption of mobile-cloud computing. Furthermore, selecting the best server(s) from among the large number of available cloud servers is also a major hurdle.

Recently, a mobile-cloudlet-cloud architecture called MOCHA (MObile Cloud Hybrid Architecture) has been developed at the University of Rochester as a framework for running computationally intensive mobile applications with low response time requirements [1], [2]. The introduction of a cloudlet, which serves as an edge-server, provides us the potential to address the hurdles for mobile-cloud computing mentioned above.

Instead of transmitting data directly from a mobile device to the cloud servers, using MOCHA, we can send the data to a cloudlet/cloudlets first. A cloudlet is capable of storing and updating a profile of the network latencies and their variation to reach different cloud servers. Using this approach, we can perform smart task division to select the best server(s) such that the overall communication latency is minimized.

The goal of this thesis is to explore the benefits in utilizing an edge-server (cloudlet) in the MOCHA architecture for mobile-cloud computing through network latency measurements, data analysis and modeling and task division algorithm simulations and validations.

1.2 Contributions

The main contribution of this thesis is the demonstration of the benefits in utilizing a cloudlet within the MOCHA architecture to support mobile-cloud computing. Both the task division simulations and the real measurements on cloudlets show that using a cloudlet provides a viable approach for selecting the best server(s), thus minimizing overall communication latency. A range of work has been done to achieve this goal.

(1) We set up the hardware platform for the MOCHA architecture. This includes two main parts: mobile device to cloudlet communication and cloudlet to cloud servers communication. For the mobile device to cloudlet connection, we establish wireless communication between an Android tablet (ARCHOS 32 internet tablet) and a PC (Window 7 64bits OS). For the cloudlet to cloud servers connections, we utilize 130 PlanetLab servers across 5 continents.

(2) We implement the software for communication between a mobile and a cloudlet, the cloudlet to the cloud servers. We develop Java code using socket packages on the mobile side and a C # program on the PC side. For the cloud

to cloud servers connection, we develop a Java program utilizing the TCP/IP protocol.

(3) We collect and process critical data for the MOCHA architecture, e.g., network latency, path route changes, etc. We solve the challenges of obtaining network information utilizing 130 PlanetLab servers across 5 continents.

(4) We analyze and model network behaviors from a number of critical perspectives: latency and its variance, latency/variance change as data size increases, latency/variance change as the number of hops increases, and communication path changes. We create latency distribution models and latency and variance linear models, which are the basis of the task division simulations.

(5) We perform task division simulations and analyze the results using different methods, including a random redundancy algorithm, a fixed algorithm and a greedy algorithm. Finally, we conduct comparisons and analysis of these methods to demonstrate their benefits.

Our results show:

Utilizing a cloudlet in mobile-cloud computing provides significant benefits (in terms of response latency) from the use of dynamic profiling (performing the fixed or greedy algorithms) for the optimal server selection. Without profiling, the random server selection is capable of providing acceptable latency performance at the cost of increased redundancy.

1.3 Organization

This thesis is organized as follows. Chapter 2 provides general background knowledge about the MOCHA architecture and PlanetLab resources. Chapter 3 introduces the experiment design of this thesis, including software and hardware development and data collection strategy. Chapter 4 presents the data analysis, including latency behavior, the latency variance behavior, and other aspects (communication path changes, latency distribution). Chapter 5 provides the task division simulations and comparisons for different task division algorithms. Finally, Chapter 6 presents the final conclusions and the future work in this area.

2 Background

In this chapter we provide general background knowledge about the MOCHA architecture and PlanetLab resources that are critical in this thesis.

2.1 MOCHA Introduction

MOCHA (MObile Cloud Hybrid Architecture) was created as a solution to support massively-parallelizable mobile-cloud applications (Soyata et al., 2012a, 2012b). MOCHA, which is shown in Figure 2.1, consists of three parts: Mobile, Cloudlet and Cloud. It formulates a solution to allow mobile-cloud computing applications, such as object recognition in a battlefield, by introducing an edge-server, which is called a cloudlet [3]. A cloudlet is used as an intermediary between the mobile devices and the cloud servers and determines how to partition the computation, in terms of tasks, among itself and multiple cloud servers to optimize the overall quality of service (QoS), based on continuously updated statistics of the QoS metrics (e.g., latency).

Running many resource-intensive applications far exceeds the capabilities of today's mobile devices, such as conducting real-time face recognition of known criminals. There are many limitations of the mobile device, such as low processor speed, small memory size, and limited storage capacity. Facing these challenges, cloud servers with tremendous computing power and storage space as well as

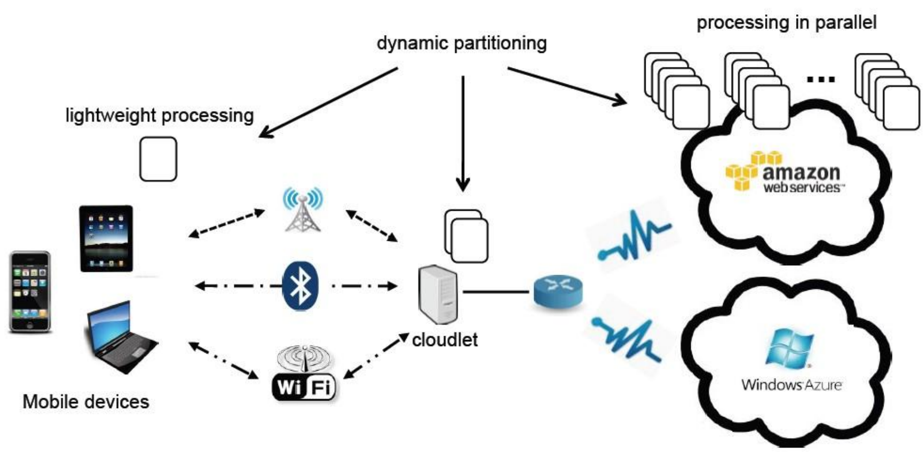

Figure 2.1: The MOCHA architecture: mobile devices interact with the cloudlet and the cloud via multiple connections and use dynamic partitioning to achieve their QoS goals (e.g., latency, cost) (reprinted from (Soyata, T., Muraleedharan, R., Funai, C., Kwon, M., & Heinzelman, W., 2012) with permission of the authors).

access to particular databases are suitable choices to accelerate applications run on mobile devices. Nevertheless, several major hurdles limit these benefits, such as long network latency, which hurts the user experience in mobile-cloud computing. As a result, powerful, well-connected and safe cloudlets are necessary to intercept the data sent from the mobile and perform smart task division algorithms to minimize the overall communication latency to and from the cloud.

There are many applications that can benefit from utilizing an edge server (cloudlet in MOCHA), such as battlefield support applications, natural language processing, airport security, an enhanced Amber Alert system, among others.

2.2 Planet-Lab Introduction

In this thesis, we utilize PlanetLab servers as cloud servers. PlanetLab is a global research network that supports the development of new network services. It was established in 2002 by Prof. Larry L. Peterson [4]. Since the beginning of 2003, more than 1,000 researchers at top academic institutions and industrial research labs have used PlanetLab to develop new technologies for distributed storage, network mapping, peer-to-peer systems, distributed hash tables, and query processing [5]. PlanetLab currently consists of 1167 nodes at 550 sites (March 18, 2013). In this thesis, we utilize 130 PlanetLab nodes across 5 continents, as shown in Figure 2.2. The geographically diverse sites in PlanetLab allow us to test different parts of the network with more comprehensive observations.

PlanetLab is a great tool to perform large-scale Internet studies. Its power lies in that it runs over the common routes of the Internet and spans nodes across the world, making it far more "realistic" than a simulation. However, this reality comes at the price of route-failures, node outages, ssh-key sharing issues, and other network realities, which make managing experiments difficult.

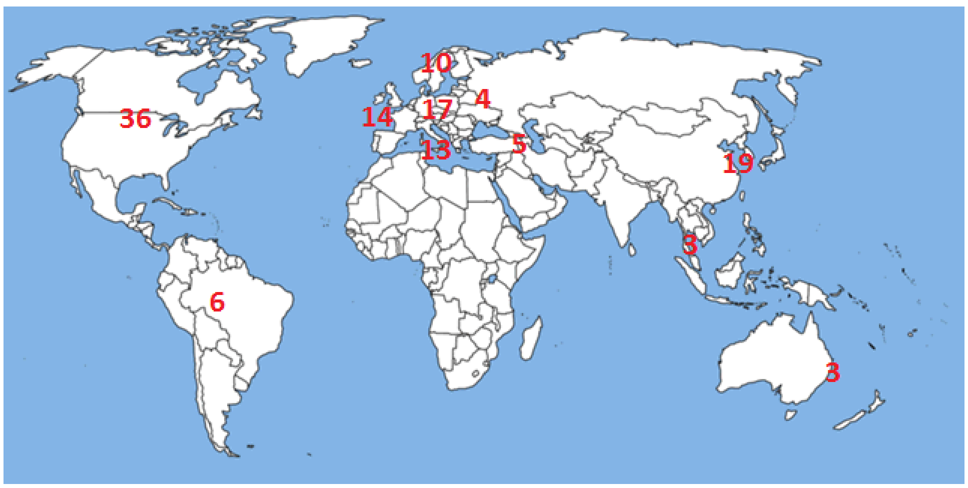

Figure 2.2: Distribution of 130 PlanetLab nodes that are used in this thesis.

3 Experiment Design

In this chapter, we present the design of our experiments, including software and hardware development as well as the data collection strategy. Details are presented for the mobile to cloudlet connection and the cloudlet to cloud connection, respectively.

3.1 Hardware and Software Development

3.1.1 Mobile to Cloudlet Connection

Wireless Connection Methods Selection

To achieve direct communication between a mobile and a cloudlet for real data measurements, we need to set up a physical communication channel. In our experiments, the cloudlet is a desktop computer (with Windows 7 64-bit operating system), while the mobile device is an ARCHOS 32 internet tablet with Android 2.2.1 OS.

There are several ways to set up a physical communication channel between a PC and a mobile device, such as Bluetooth, WiFi and 3G. In this thesis, we choose WiFi because it is the most commonly used technology for wireless communication between a mobile and a PC. There are two approaches to achieve wireless communication between the PC and the Android tablet via WiFi.

(1) Set up a wireless ad hoc network

A wireless ad-hoc network is a decentralized type of wireless network. The network is ad hoc because it does not rely on a pre-existing infrastructure, such as routers in wired networks or access points in managed (infrastructure) wireless networks. Instead, each node participates in routing by forwarding data for other nodes, and so the determination of which nodes forward data is made dynamically based on the network connectivity [6].

(2) Set up a wireless access point network

In computer networking, a wireless access point (WAP) is a device that allows wireless devices to connect to a wired network using WiFi, Bluetooth or related standards. The WAP usually connects to a router (via a wired network), and can relay data between the wireless devices (such as computers or printers) and wired devices on the network [7].

Internet access via ad hoc networks, using features like Windows Internet Connection Sharing, may work well with a small number of devices that are close to each other, but ad hoc networks do not scale well. Internet traffic will converge to the nodes with direct Internet connections, potentially congesting these nodes. For Internet-enabled nodes, access points have a clear advantage, with the possibility of having multiple access points connected by a wired LAN. Furthermore, the Android devices do not support ad hoc mode automatically. Consequently, in this thesis, we decided to set up a wireless access point (WAP) at the PC (cloudlet) so that the Android tablet (mobile device) can connect to the cloudlet through the WAP.

Wireless Access Point Setup

In the Window 7 operating system, there is a new function called Microsoft Virtual WiFi. Based on one real WiFi card, Window 7 has the capability to create a virtual WiFi card, which can be used to set up a wireless access point. This new feature provides us a way to achieve the physical communication between the cloudlet and the mobile device with minimum cost. The basic procedures for setting up the connection in this way are as follows:

Step 1: Run cmd.exe with Administrator Privileges

The access point setup must be under administrator privileges. In the "Start" menu, type "cmd" then right click the search result, choose "run as administrator" or with hotkey combo "Ctrl+Shift+cmd.exe."

Step 2: Start and Set the "Virtual WiFi" Mode

Using command lines: *Netsh wlan set hostednetwork mode=allow ssid=TestAP001 key=123456789* to set up the access point, Figure 3.1 shows the results. Here "mode" determines whether the Virtual WiFi is started or not; "ssid" is the name of the wireless network and "key" is the password of the wireless network.

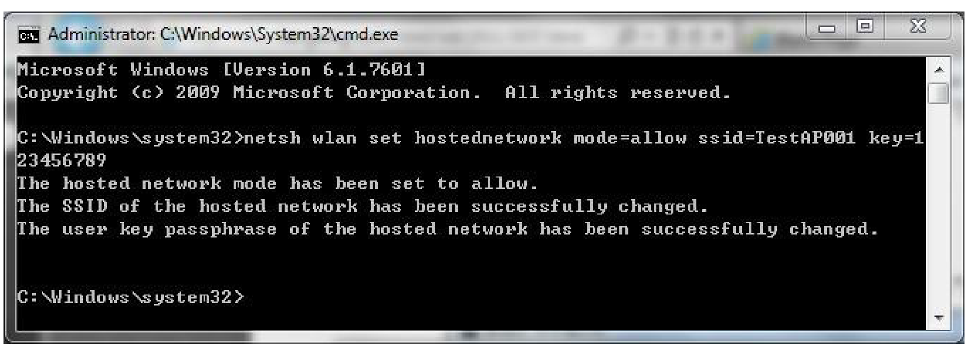

Figure 3.1: Setup information of the wireless network.

Step 3: Start Internet Connection Sharing (ICS)

This step is not necessary for the PC-Mobile connection setup. It is useful only if one also wants to use the network for internet surfing. Open "internet connection," choose "change the adapters setting," and right click one adapter with internet connection, then choose "properties." In the menu "share," choose "TestAP001" as the target to share the Internet. Figure 3.2 shows the setup information.

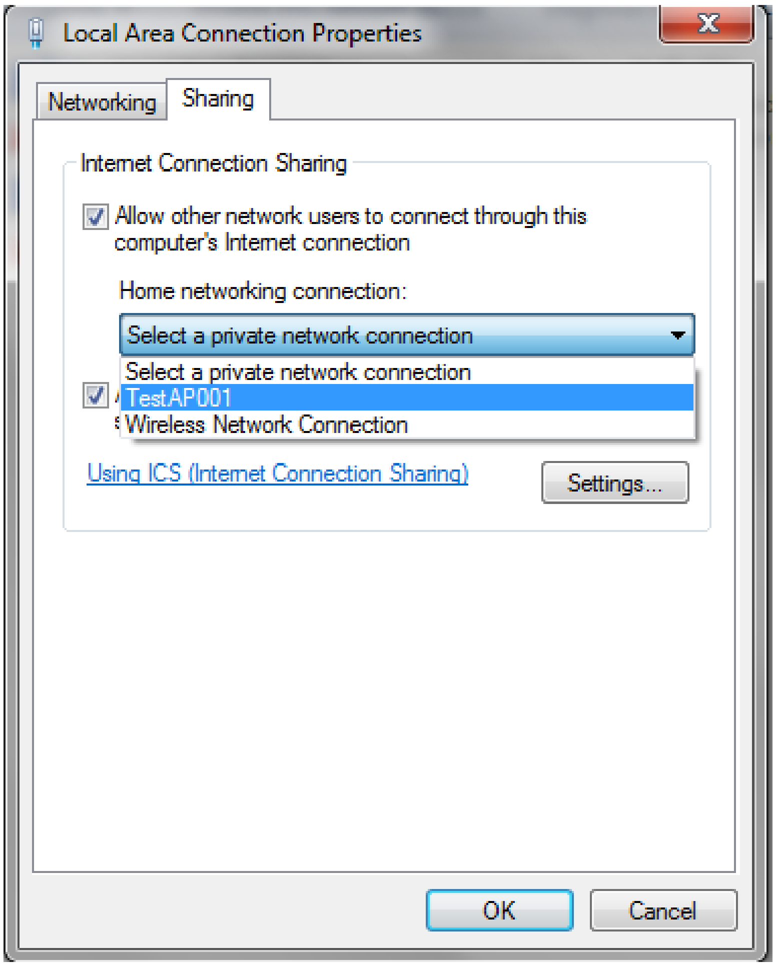

Figure 3.2: Internet connection sharing setup information.

Step 4: Start the Wireless Network in the Command Line Window Using the command line: *Netsh wlan start hostednetwork* to start the access point, Figure 3.3 shows the corresponding information.

After these 4 steps, the wireless access point is successfully set up. The Android tablet is able to find this network and join it. Moreover, through the IP address

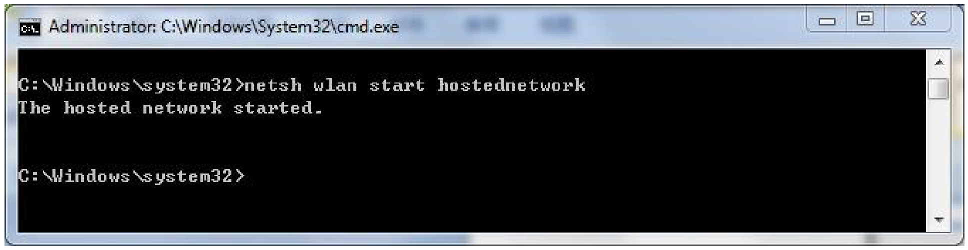

Figure 3.3: Command line for starting the wireless network.

of the access point, the mobile can send data directly to the PC (cloudlet) and the PC (cloudlet) can send data directly to the mobile.

Transmitter and Receiver Program Development

After setting up the access point on the PC (cloudlet), we develop the transmitter program (mobile device side) in Eclipse Java EE IDE using socket packages and the receiver program (cloudlet side) in Microsoft Visual Studio 2010 using Window Sockets API (WSA).

The Windows Sockets API (WSA), which was later shortened to Winsock, is a technical specification that defines how Windows network software should access network services, especially TCP/IP. It defines a standard interface between a Windows TCP/IP client application and the underlying TCP/IP protocol stack [8]. In addition, TCP/IP provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed and received at the destination [9].

The mobile transmitter program (Colin Funai, 2012) was first developed by Colin Funai of the MOCHA research group. We modify this program to send different sizes of data via the wireless network. Figure 3.4 shows the application user interface at the mobile. The cloudlet receiver program is capable of receiving data from the mobile and sending back acknowledgment messages. Figure 3.5 shows the user interface at the cloudlet.

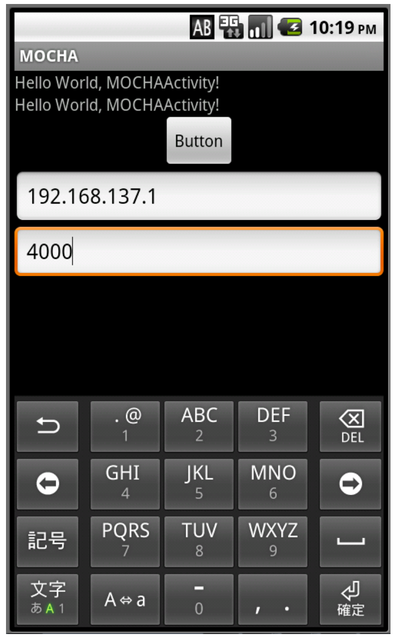

Figure 3.4: Transmitter program user interface at the mobile.

3.1.2 Cloudlet to Cloud Connection

As we mentioned in Section 2.2, we utilize PlanetLab nodes as cloud servers in this thesis.

PlanetLab is a group of computers available as a testbed for computer networking and distributed systems research worldwide. Each research project has a “slice,” or virtual machine access to a subset of the nodes. Figure 3.6 shows the “slice” information for the work described in this thesis, where U is the number of users and N is the number of nodes.

PlanetLab Setup

Computers on PlanetLab are called nodes. These nodes can only be interacted with by using the OpenSSH protocol (i.e., SSH). Furthermore, they can only be accessed via a public/private key pair. To create a public/private key pair, we use the following command:

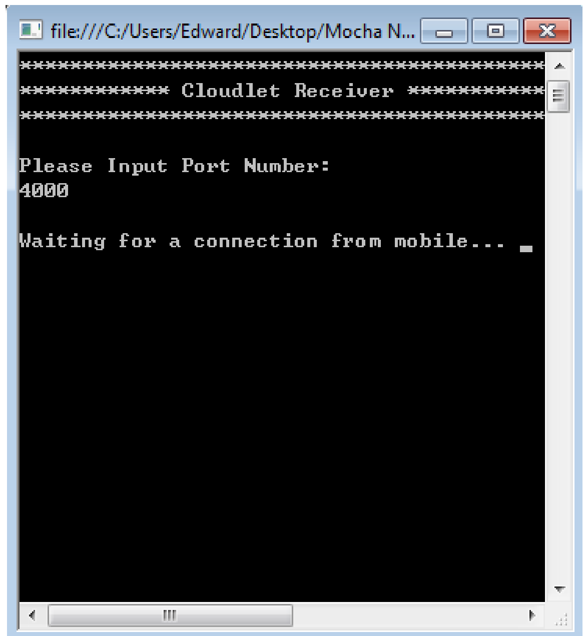

Figure 3.5: Receiver program user interface at the cloudlet.

PEER	NAME	USERS	U	N	EXP. D/M/Y
PLC	urochester_mocha	zdou2@z.rochester.edu	3	198	01/04/2013

Figure 3.6: PlanetLab “Slice” information of the work described in this thesis.

sshkeygen -t rsa -f planetlab-key

Then a password will be created. This password is used to secure the private key and prevents others from using your slice.

After the above command runs, a planetlab-key file will be generated (encrypted private key). planetlab-key.pub is the public key and needs to be uploaded to the PlanetLab account on planet-lab.org. PlanetLab then distributes this public key to all the nodes in the UR MOCHA slice.

To connect to a Planet-lab node the following syntax is used:

ssh -i planetlab-key slicename@hostname

Then the user must enter the private key password. If the connection is refused, it may be that node is off-line currently. It is cumbersome (yet secure) to type in the ssh-key password, especially when performing experiments associated with a large number of nodes and over a long period of time. To login automatically without entering in a password, OpenSSH has a program called ssh-agent, which allows private keys to be decrypted and stored in memory for password-less logins. An ssh-agent can be run using the command:

eval 'sshagent'

ssh-add planetlab-key

This then only requires the user to type in the password once for the private key, and once this is entered, using ssh to log into the nodes does not require the -i switch or entering in the password.

The first time connecting to a node via SSH, OpenSSH asks the user whether or not the user wishes to accept the nodes' public key. The user has to type "yes" in order to authenticate the node. To eliminate the key check, the user must add a file called "config" to the folder $HOME/.ssh folder. This file contains the line:

StrictHostKeyChecking no

Client and Server Program Development

For the client transmitter and server receiver program, we develop Java programs using sockets. A socket is a dual-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes: Socket and ServerSocket to implement the client side and the server side, respectively.

The server-side program is started using the command:

Java Server001 port-number data-size repeat-count

The receiver program is capable of receiving arbitrary size data and sending back an acknowledgment message via the same connection.

The client-side program is started using the command:

Java Client001 slicename@hostname port-number data-size repeat-count

The transmitter program has the capability of sending different sizes of data to the servers and recording the time used after receiving the acknowledgment message from the server.

In order to run the client and server Java programs in the PlanetLab nodes in our Slice, we need to install the Java run time environment on the PlanetLab nodes. This is done using the following command:

sudo -c "yum install java-version-gcj-compat-devel"

For monitoring and organizing the client program, server program and the PlanetLab nodes' information together, we utilize Shell scripting in Linux. Writing a shell script is much quicker than writing the equivalent code in other programming languages, and there are many advantages, including easy program or file/programs selection, quick start, and interactive debugging.

The most common Linux shell is named “Bash” (Bourne Again SHell). The followings commands show an example bash shell program used in this thesis.

```
#!/bin/bash
cat nodesClient.txt | while read LINE
do set −− $LINE
xterm -hold -e “ssh urochester_mocha@ $1 ‘bash runClient.sh’ ”&
done
exit
```

This shell program is used to automatically start a client program in all client nodes specified in *nodesClients.txt*. Using the xterm command we can create a monitoring window for printing out the debugging information of the running client program. Moreover, as we can see in this shell program, another shell program, *runClient.sh*, is embedded into the current one. The *runClient.sh* is used to manage multiple client programs with different specifications and constraints. It is convenient to embed shell programs together to perform sophisticated scheduling and achieve automatic operation.

3.2 Data Collection Strategy

In this section we introduce the data collection strategy for the mobile to cloudlet communication link and the cloudlet to cloud communication link. The primary information we measured is the two-way communication latency, also called the response latency (i.e., the time between when one client starts sending a certain size of data and the client receives an acknowledgement message sent from the destination server via the same connection).

Name	Parameters
Time	Weekdays
Data Size	8,64,256,512,1024,2048,5120,10240 (KB)
Data Points	500 per data size
Category	One-hop and Multi-hop WiFi
Hardware	ARCHOS 32 internet tablet
	PC with Window 7 64bits OS

Table 3.1: Mobile to cloudlet measurement parameters.

3.2.1 Mobile to Cloudlet Latencies

Our first set of measurements is the two-way communication latencies from the Android-based mobile device to the cloudlet (PC) via 1) a direct WiFi connection as described in Section 3.1.1 and 2) a WiFi hotspot. The connection is a single hop in the former and multiple hops in the latter. The parameters for this set of measurements are shown in Table 3.1.

3.2.2 Cloudlet to Cloud Latencies

For the cloudlet to cloud measurements, we once again collect the information about the two-way communication latencies. In addition, we also record the number of hops in the path between the client (cloudlet) and the servers using the traceroute command in Linux. The parameters for this set of measurements are shown in Table 3.2.

Name	Parameters
Time	Weekdays, Weekends
Data Size	8,64,256,512,1024 (KB)
Data Points	400 per data size
Client Number	5
Server Number	130
Category	Wide-area networks (WAN)
Hardware	PlanetLab nodes

Table 3.2: Cloudlet to cloud measurement parameters.

4 Data Analysis

In this chapter, we present data analysis for the measured latencies for data transfer between the mobile and the cloudlet and between the cloudlet and the cloud. We examine the average latency as well as the variance in latency, as a function of the data size and the number of hops in the route. The motivation is to study the capabilities of the cloudlet in order to minimize the overall communication latency through intelligent server selection when abundant cloud resources are available. In particular, we want to determine whether we can use the measured latencies to create models for estimated latencies for different cloud servers. This "profiling" would enable the cloudlet to select the server(s) that would minimize overall latency. We will see in the next chapter the benefits of such an approach.

4.1 Latency Measurements

For the mobile to cloudlet connection, Figures 4.1 and 4.2 show the maximum, mean, and minimum latencies as the file size changes from 8 KB to 10 MB for one-hop and multi-hop WiFi, respectively. For a given data size, the highest point indicates the maximum, the lowest is the minimum, and the middle one with an error bar denotes the mean latency with standard deviation (error bar).

For the cloudlet to cloud connection, Figures 4.3 and 4.4 show the maximum, mean, and minimum latencies as the file size changes from 8 KB to 1 MB for 2

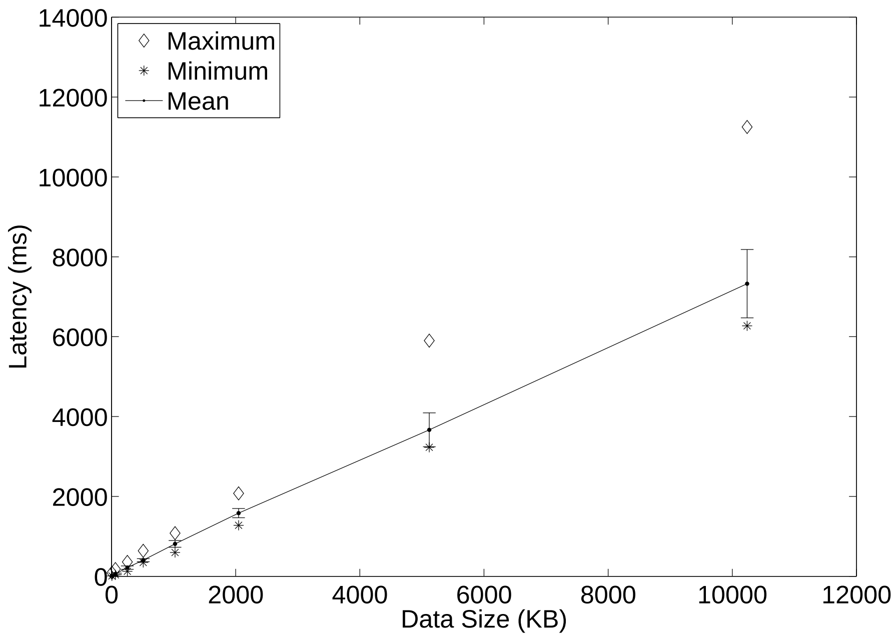

Figure 4.1: The mean, maximum, and minimum latencies as the file size changes for one-hop WiFi.

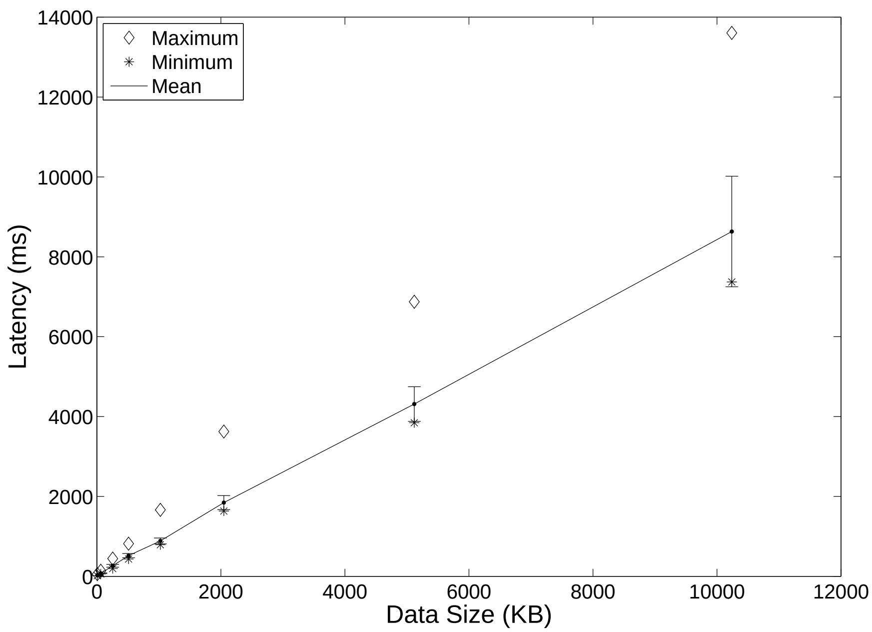

Figure 4.2: The mean, maximum, and minimum latencies as the file size changes for multi-hop WiFi.

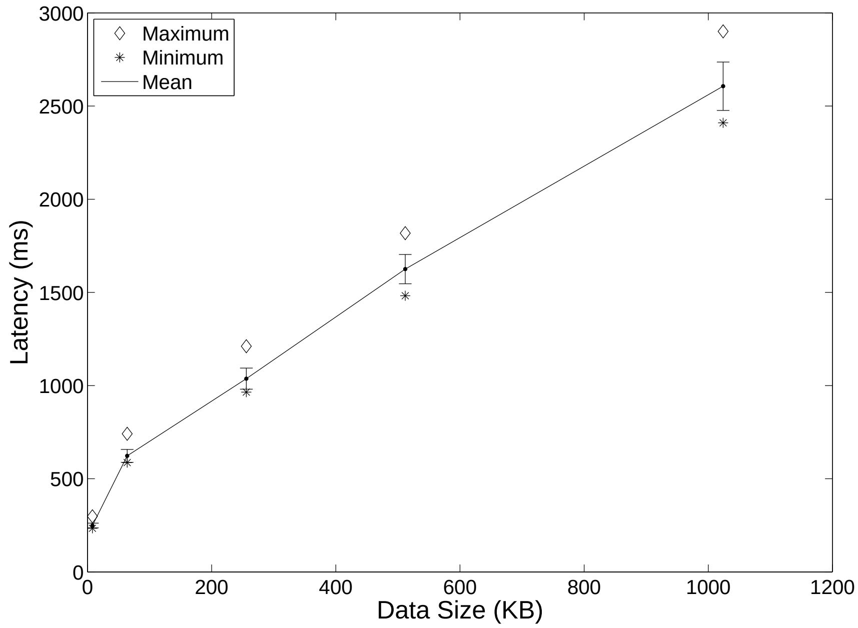

Figure 4.3: The mean, maximum and minimum communication latencies as the file size changes for communication over Planet-Lab nodes with the client in the US and the server in Europe.

example server nodes (the client in the US, the server in the US for one and the server in Europe for the other) among the 130 nodes.

The results clearly indicate that the difference between the maximum and minimum latencies increases with file size for both the single hop and multi-hop cases as well as for the PlanetLab nodes. Furthermore, the data also show that the mean latency increases with data size in a linear fashion. This feature can be utilized to create a latency model for different nodes. These latency models can be stored and updated in the cloudlet in order to optimize the server selection and minimize the overall communication latency, as we will see in the next chapter.

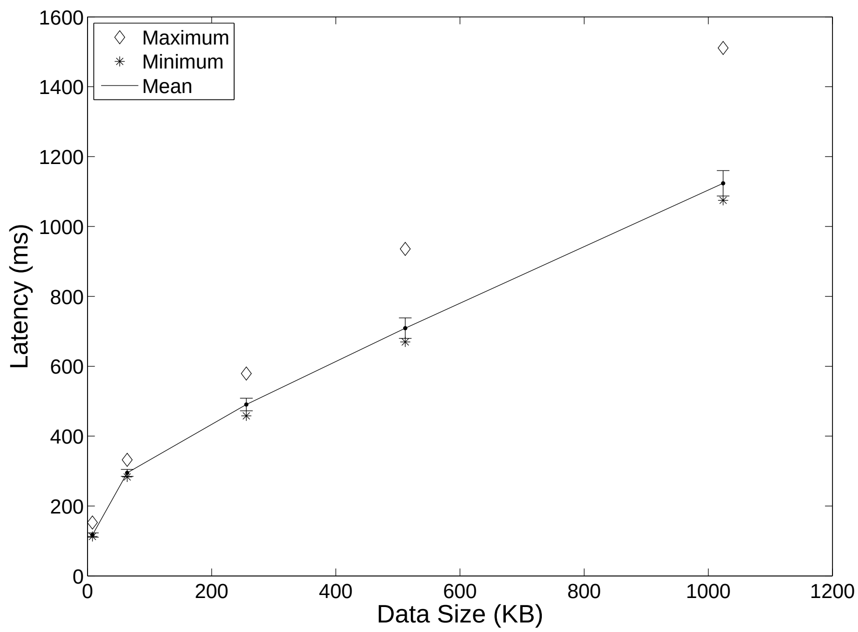

Figure 4.4: The mean, maximum and minimum communication latencies as the file size changes for communication over Planet-Lab nodes with the client in the US and the server in the US.

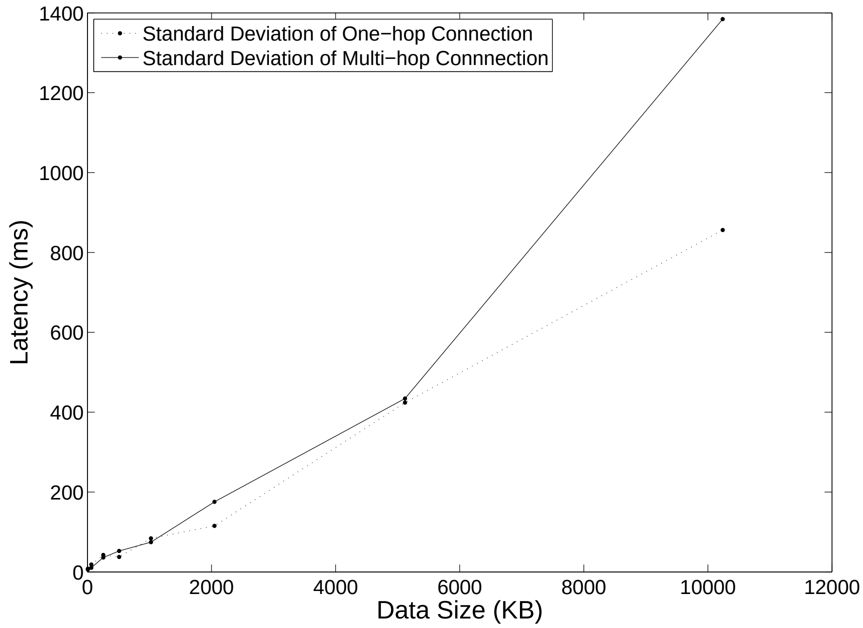

Figure 4.5: Standard deviations of communication latency via single hop and multi-hop WiFi connections.

4.2 Latency Variance Behavior

For the mobile to cloudlet communication, Figure 4.5 shows the standard deviation as the file size changes from 8 KB to 10 MB for one-hop and multi-hop WiFi. The results clearly indicate that the standard deviation increases with file size for both the single hop and multi-hop connections.

For the cloudlet to cloud communication, Figure 4.6 shows the standard deviations of latency for 130 nodes with the same average latency when 1 MB data is transferred. The data in Figure 4.6 indicates, roughly speaking, that there is a monotonic trend that latency variance increases as the latency increases. This trend indirectly suggests that there is potential to use this feature for controlling client-server pairs to optimize the server selection, especially when there are plenty of diverse cloud resources available.

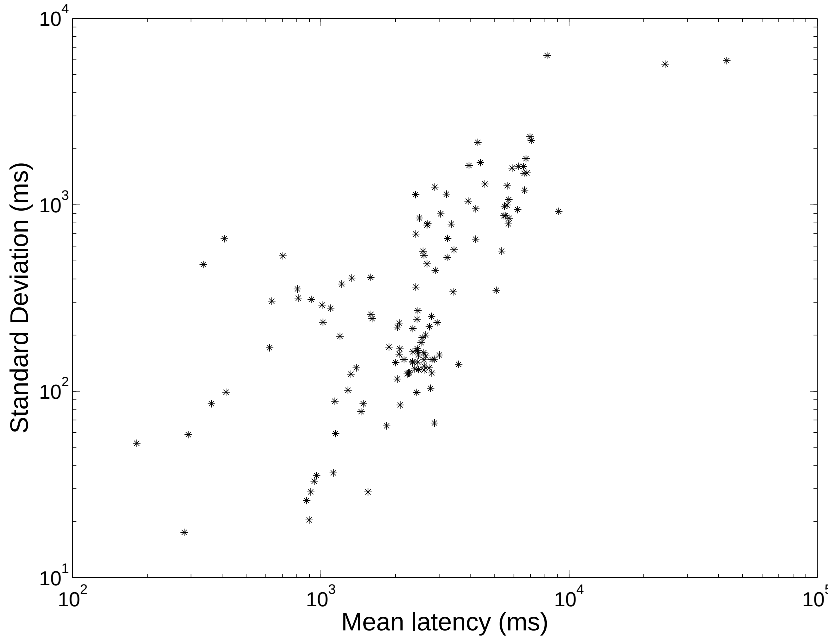

Figure 4.6: Standard deviations of communication latency for different nodes with the same average latency when 1 MB data is transferred.

4.3 Other Related Aspects

4.3.1 Communication Path Changes

In [10], the author shows that approximately 68% of the routes in the Internet do not change for seven days, while the rest change in the range of between a second and 6+ hours.

In Sections 4.1 and 4.2, we have studied the communication latency and its variance behavior with the measurements being performed for both weekdays and weekends. We aim to figure out if there is a frequent path change during the measurement time. If the path changes frequently, it is difficult and meaningless to model the latency for different servers. Table 4.1 shows the trace-route results for ten randomly chosen servers. Table 4.2 shows the measurement parameters.

Our experiment corroborates the observations of [10]. Only one pair out of ten changes its path and the number of hops several times, and all of the other pairs never change their paths. This feature ensures our basis for the simulations that will be introduced in Chapter 5.

4.3.2 Latency Distribution

Figures 4.7 and 4.8 show histograms of the latencies for one-hop and multi-hop connections for 10MB data transfer, respectively. Figure 4.9 shows a histogram of the latencies for 1MB data transfer of one example server node in PlanetLab. The latency distributions (666 distributions in total) have the same behavior for all file sizes, and they are omitted here.

All histograms indicate an identical shape, which is similar to a Rayleigh distribution (with an x-axis shift) with a peak close to the average and then gradually decreasing as latency increases. This feature can be used to build a

Node#	Node Area	Path Change	# of Hops
1	North America	No	14
2	North America	No	14
3	North America	No	15
4	Asia	No	14
5	South America	No	14
6	Europe	No	15
7	Europe	Yes	19(17)
8	Oceania	No	15
9	Asia	No	17
10	Asia	No	14

Table 4.1: Traceroute results for ten randomly selected servers.

Name	Parameters
Time	Saturday (pm), Sunday (am & pm), Monday (am & pm)
Data Size	8,64,256,512,1024 (KB)
Data Points	500 per data size
Category	Wide-area networks (WAN)
Hardware	PlanetLab nodes

Table 4.2: Path changes measurement parameters.

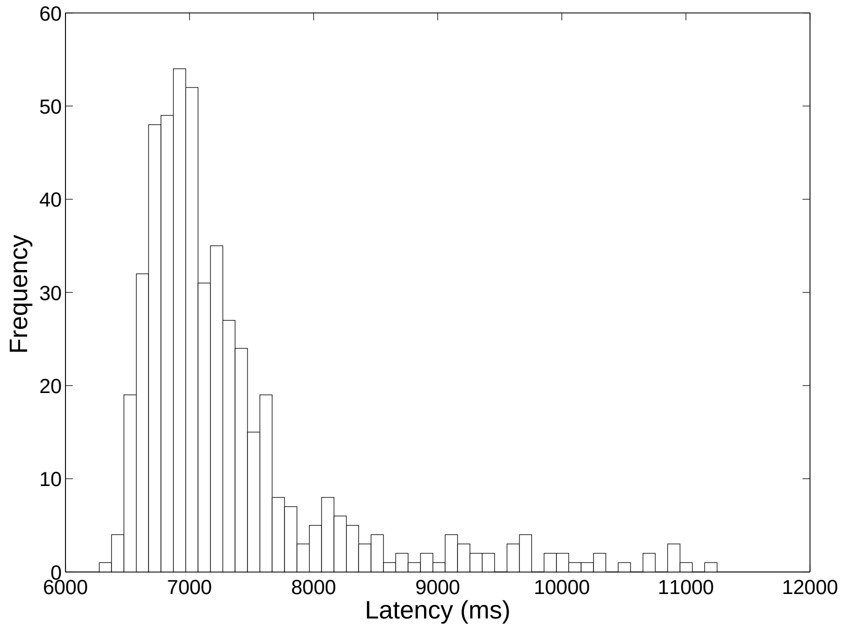

Figure 4.7: Latency distributions with 10MB data transfer for one-hop WiFi.

latency model for each data size, which could be stored and updated in the cloudlet for optimizing the server selection in order to minimize the overall communication latency.

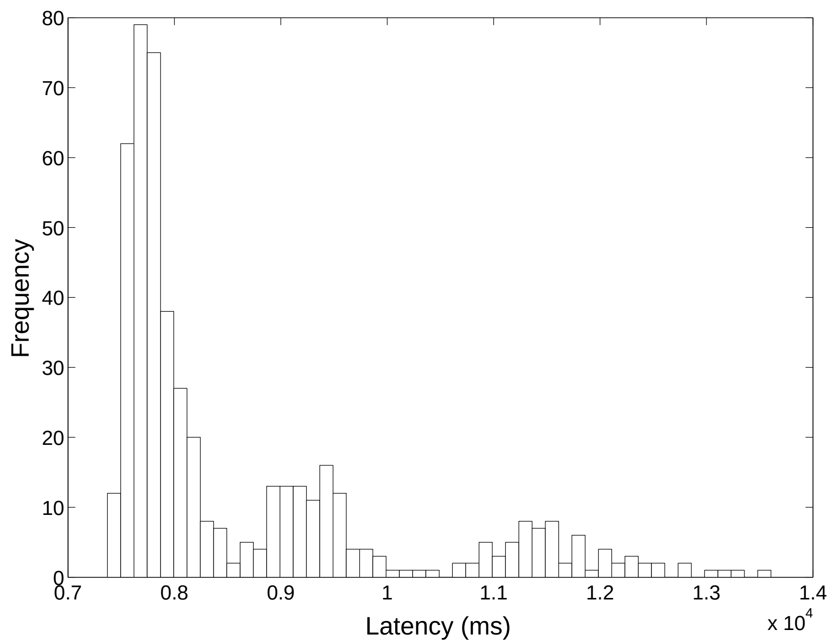

Figure 4.8: Latency distributions with 10MB data transfer for multi-hop WiFi.

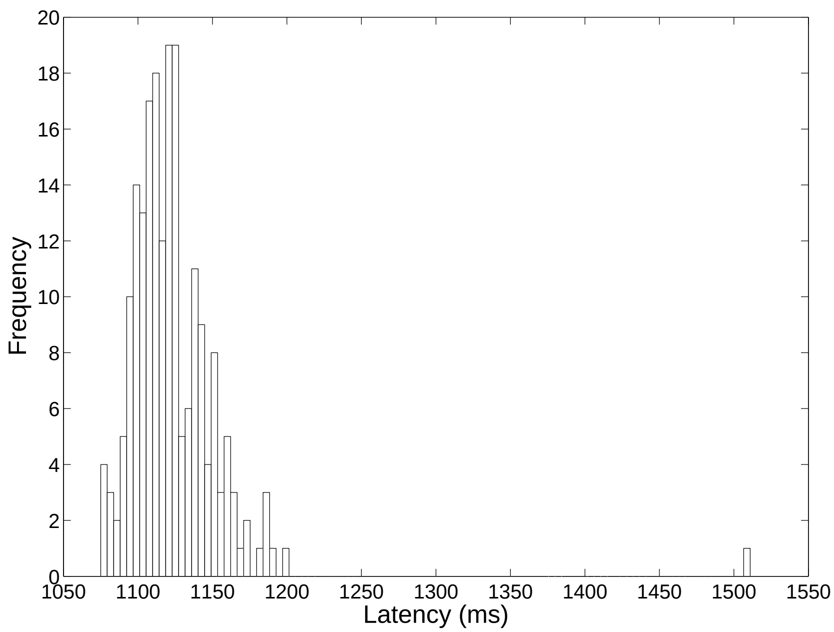

Figure 4.9: Latency distribution with 1MB data transfer for communication between a pair of PlanetLab nodes.

5 Task Allocation Techniques: Analysis and Evaluation

In this Chapter, we present several techniques for task allocation: random allocation, fixed allocation and greedy allocation. We simulate these different algorithms and conduct real measurements using the different algorithms to evaluate the benefits and tradeoffs in terms of latency and overhead.

5.1 Latency Linear Model

Using a cloudlet enables the use of smart allocation algorithms to minimize the overall communication latency to the cloud server(s). If the cloudlet has a-priori information about the latencies (and variances) to different cloud servers, it can optimize the performance. The cloudlet is able to update this information on a fairly regular basis to ensure accuracy.

As we have shown in Chapter 4, based on the measurement results, the latency displays a linear behavior as a function of data size, roughly speaking. Also, the latency distribution for a certain data size is similar to a Rayleigh distribution and can be approximated by a Gaussian distribution for simplicity.

Consequently, we create two models to support the task allocation algorithms: (1) latency linear model; (2) latency standard deviation linear model. With these two models, when given the latency distribution model (Gaussian model used in this thesis), we are able to predict communication latency given a certain server node and a certain data size with a certain probability. The latency distribution models are used to perform Monte-Carlo simulation for task allocation algorithms.

Figures 5.1, 5.2 and 5.3 show the linear regression plots for three different server nodes in PlantLab, including Instituto Tecnologico Buenos Aires, Zuse Institute Berlin and University of Oregon. All of them are receiving data from a University of Rochester node. We model these latencies as $y = ax + b$, as shown in Table 5.1, where x is the data size and y is the latency.

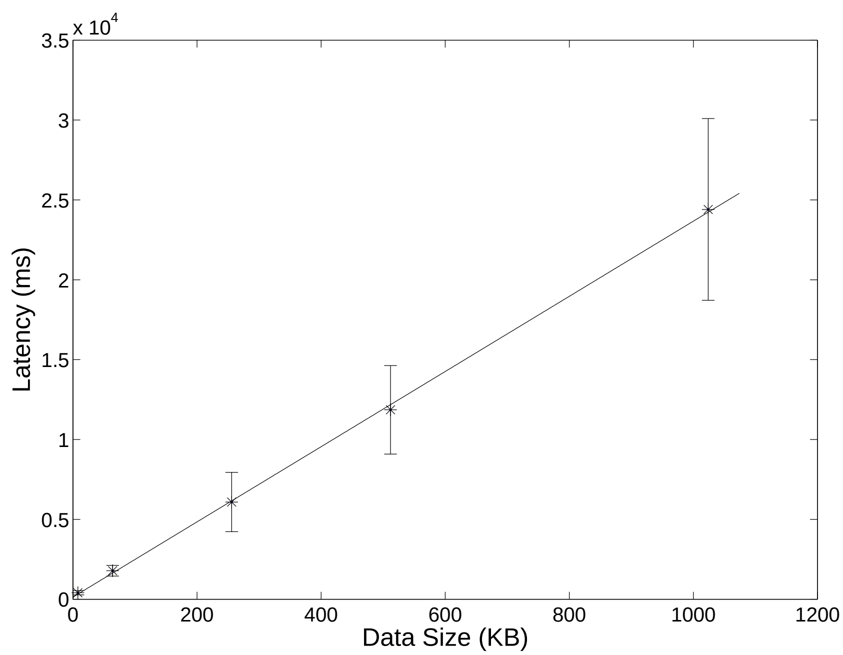

Figure 5.1: Latency linear regression for Instituto Tecnologico Buenos Aires.

These models will be used to estimate latency to a target server for an arbitrary data size when the original data is split into multiple chunks and sent to multiple servers. For example, when we send 1MB data to 3 servers, the data are split into three 341KB data chunk, approximately. Then we use the latency and variance

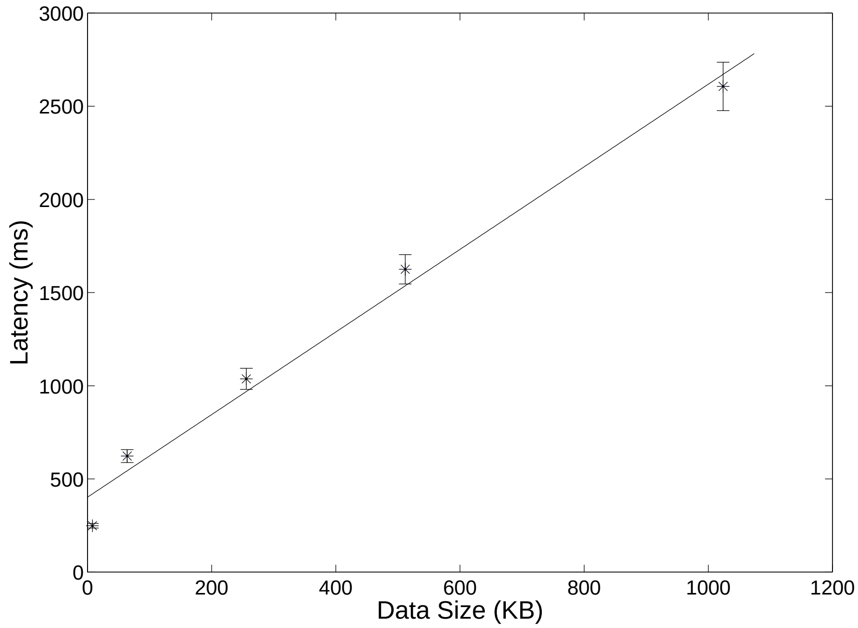

Figure 5.2: Latency linear regression for Zuse Institute Berlin.

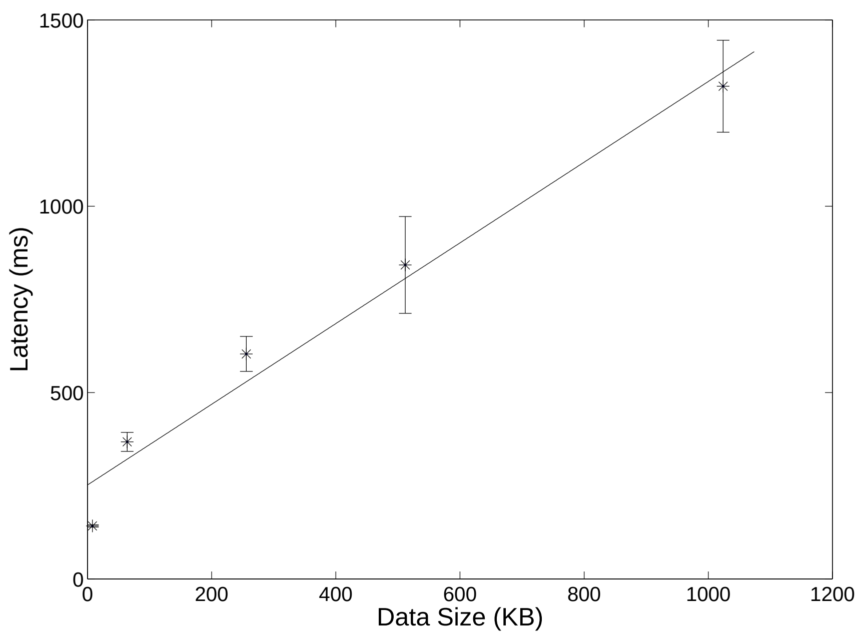

Figure 5.3: Latency linear regression for University of Oregon.

Server	a	b
Buenos Aires	23.5	13.3
Berlin	22.2	40.2
Oregon	10.8	25.2

Table 5.1: Linear model $y = ax + b$ (in ms) for servers in Buenos Aires, Berlin, Oregon.

linear models to predict the latency and standard deviation for transferring 341KB data.

5.2 Server Selection Algorithms

For our experiments, we assume that a task must be performed on a large amount of data. The task is performed on each data item independently. As an example, suppose a face recognition algorithm must be performed on 1000 faces. All 1000 faces can be sent to one server for computation, or each face can be sent to 1000 different servers (or any combination in between). Thus, the client has the option to divide the full data set into smaller chunks and send these chunks to different servers for computation. The servers return the result of the computation, and this is aggregated to obtain the final result. The question we address here is: what is the best way to divide the data among different servers? We answer this by evaluating different data division algorithms through simulations using the latency and variance models and through experiments on PlanetLab. In addition, the latency standard deviation linear model and latency distribution model are only required in the simulations. For the real cloudlet, for now, the profile is only the mean of the two-way communication latency of different network links/routes.

5.2.1 Random Algorithm

For the random algorithm, a client sends M bytes of data to N randomly selected servers with different communication latency models. The M bytes of data are evenly divided into N tasks with $\lceil \frac{M}{N} \rceil$ bytes of data sent to each of the N servers. Then, the client aggregates the N responses received from the N servers. We choose the final response received as the overall latency for all M bytes of data.

Furthermore, the client may send copies of the data to more than one server for redundancy purposes. For example, client A sends 3 copies of data chunk 1 to 3 different servers, and client A is capable of receiving 3 responses for data chunk 1 due to this redundancy operation. In this case, we choose the first response received for data chunk 1 (with minimum latency) as the latency for data chunk 1. Similarly, we choose the last response received for all the sub-tasks as the total latency for completing the task.

The use of redundant servers for the same data chunk is expected to reduce the overall latency by increasing the probability that a response for a given data chunk is returned earlier. Figure 5.5 shows the simulation results for the random algorithm with 1, 3, 5 and 10 redundant copies of each data chunk sent to different servers, as the number of data chunks (x-axis number of servers) is increased. Hence, when the x-axis "number of servers" is equal to 5, this means the original data is split into 5 chunks. For a redundancy of 10 (R10 in the figure), each of these 5 data chunks are sent to 10 different servers, hence in all 50 servers are used to process the 5 chunks of data. Figure 5.4 shows the real measurement results for the random algorithm with the same parameters of the simulation.

As we can see from Figure 5.5, the latency decreases as redundancy increases (R3, R5, and R10) since more copies (e.g., more servers) are selected and the probability of receiving an earlier response increases. This shows the benefit of using a higher amount of redundancy in terms of latency. Obviously, the trade-off

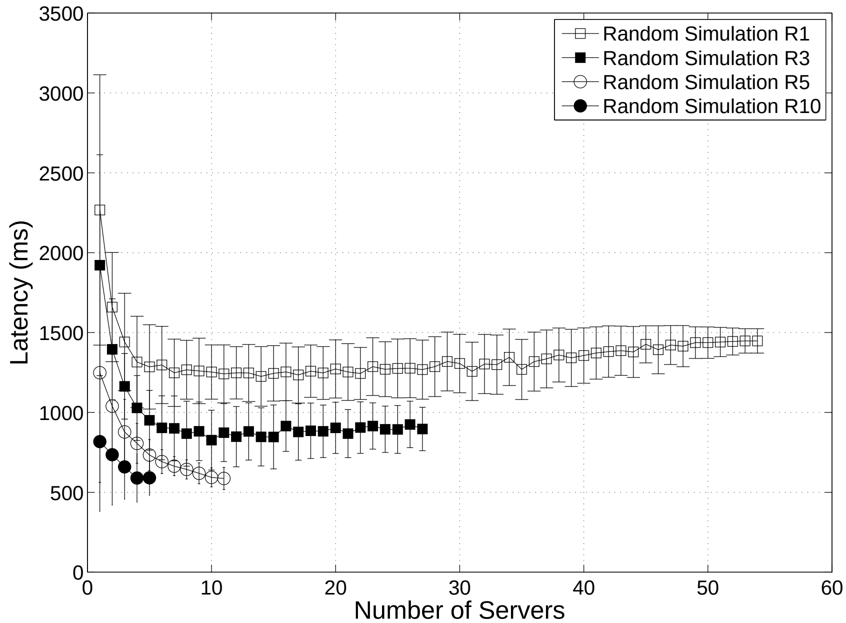

Figure 5.4: Simulation: the random algorithm with 1, 3, 5 and 10 redundant copies of the data sent to the same number of servers. Here, R1 means that there is only one copy of the data sent to the servers; R3 means there are 3 copies of the same data sent to three different servers; R5 means there are 5 copies of the same data sent to five different servers; and R10 means there are 10 copies of the same data sent to ten different servers. The x-axis represents the number of servers utilized for processing when there is no redundancy. So, for example, at an x-axis point of 5 servers, this means the original data is broken into 5 chunks and each chunk is then sent to $R = 1, 3, 5, 10$ different servers for processing. Given our overall limited number of servers, at a redundance of $R = 10$, we do not have enough servers to split the data into more than 5 chunks (which translates to 50 servers). Here, data size M=1024 (KB), total number of servers = 55.

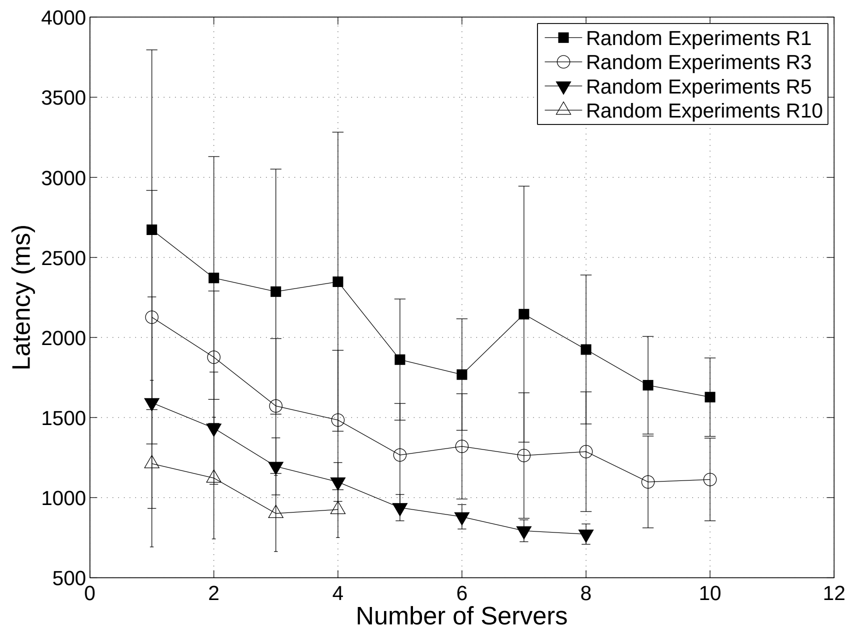

Figure 5.5: Experiment: the random algorithm with 1, 3, 5 and 10 redundant copies of the data sent to the same number of servers. Given our overall limited number of servers, at a redundance of $R = 10$, we do not have enough servers to split the data into more than 4 chunks (which translates to 40 servers). Here, data size M=1024 (KB), total number of servers =41 (41 of 55 nodes with valid data).

is the enormous overhead in terms of redundant data sent through the network and redundant computations performed on that redundant data by the servers.

5.2.2 Fixed Algorithm

The fixed algorithm is the same as the random selection algorithm except for the following two aspects: (1) there is no redundancy operation; and (2) profiling of the latency information is required. When sending $\lceil \frac{M}{N} \rceil$ bytes of data, for the fixed algorithm, we sort the servers in ascending order in terms of the average estimated latencies based on the latency linear model for $\lceil \frac{M}{N} \rceil$ data size. Then we select the first N severs from the lowest latency upward. The ultimate communication latency is the time that it takes for the last response to arrive from these N selected servers. Figure 5.6 shows the simulation results and the real measurement results for the fixed algorithm.

As we can see from Figure 5.6, the latency decreases initially and increases after a certain number or more severs (e.g., 5 in Figure 5.6) are involved. Latency initially decreases since the data is split into more chunks and the data chunks are sent to low latency servers. After a certain number of servers is reached, servers with high latency must be used so that the burden sharing benefit is overshadowed by the long latencies to these servers.

5.2.3 Greedy Algorithm

For the greedy algorithm, like the fixed algorithm, we order the servers by their average estimated latencies based on the latency linear model. Then we give the first chunk of data (here the chunk of data means the smallest chunk that a task can be divided into) to the server that can complete this task in the minimum amount of time. Then, we assign the second chunk of data to the server that

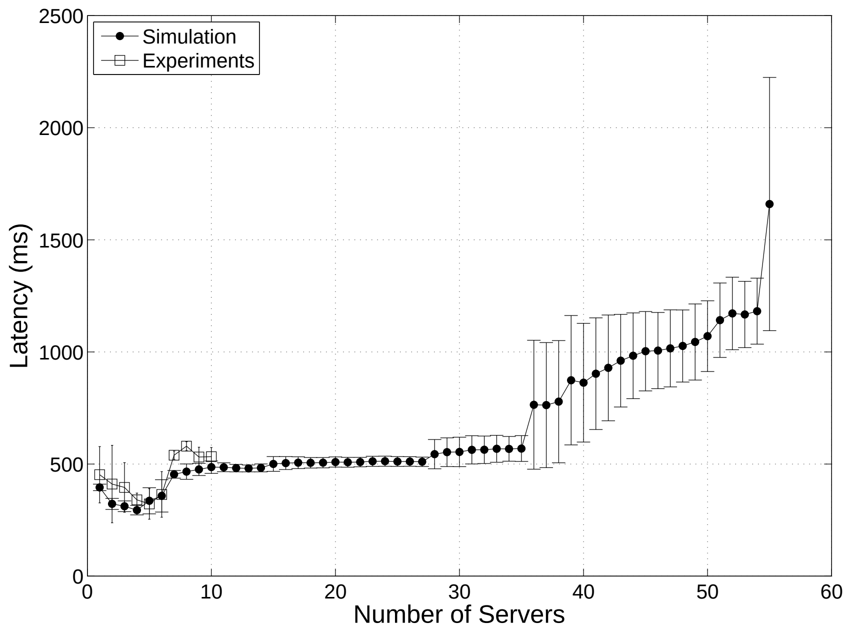

Figure 5.6: Simulation and experiment results for the fixed algorithm as we increase N, the number of servers considered for sending data. In this algorithm, each server receives $\lceil \frac{M}{N} \rceil$ amount of data. Here, data size M=1024 (KB).

can complete this task in the minimum amount of time (note that this may be the same server as given the first task if the time for the first server to complete both tasks 1 and 2 with data chunks 1 and 2 is less than the time for the second server to complete just task 2 given only data chunk 2). We continue in this way, greedily selecting the server for each task in turn. Finally, a set of servers with a different number of tasks/data chunks are determined. The overall response time of the greedy approach is the time it takes for the last response to be received from all servers.

Finding this optimal set of such data chunks is inherently a complicated optimization problem, we use the approximation algorithm described in Algorithm 5.2 for our simulations and experiments. The main idea is to move a portion of data ($a(i) * u$, where $a(i)$ is the latency linear model slope for server i and u is the unit data chunk) from server i when $i = 1 : k - 1$ to a newly added server k, and iterate the process until finding the minimum latency.

Figure 5.7 shows the simulation results and the real measurement results for the greedy algorithm. As we can see from Figure 5.7, the latency decreases and becomes stable after a certain number of servers are added. The greedy algorithm aims to find the optimal server selection with minimum latency. At the stabilized point, high latency servers (with high fundamental overheads) would be required if the number of servers was increased, and this does not provide any benefit for reducing latency. As a result, these servers will not be selected and the set of selected servers remains the same as before.

5.3 Comparison Among the Algorithms

The cloudlet is capable of storing and updating the network latency profiles, which makes the fixed and greedy algorithms possible for performing optimal server

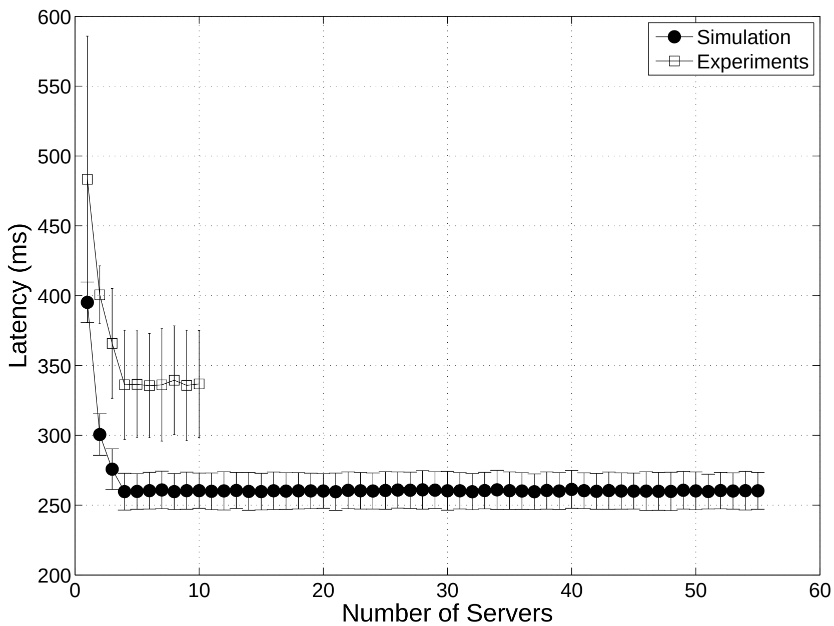

Figure 5.7: Simulation and experiment results for the greedy algorithm as we increase N, the number of servers considered for sending data. In this algorithm, each server receives a different amount of data depending on what data size at that server minimizes latency. Here, data size M=1024 (KB).

for $k = 2 : n$ do
$flag = true$
while $flag$ do
for all $server(i)$ as $i = 1 : k - 1$ do
$temp(i) = data(i)$
$data(i) = data(i) - a(i)u$
end for
$data(k) = totalData - \sum_{k-1}^{i=1} data(i)$
$curLat = \infty$
if $max(lat(0), lat(1), ..., lat(k)) < curLat$ then
$flag = true$
$curLat = max(lat(0), lat(1), ..., lat(k))$
else
$flag = false$
for all $server(i)asi = 1 : k - 1$ do
$data(i) = temp(i)$
end for
end if
end while
end for

$data(i)$: data size sent to $server(i)$

$a(i)$: rate that latency increases as $data(i)$ increases

$lat(i)$ latency of $server(i)$

u unit data chunk

$totalData$: total data size

Table 5.2: Approximated Greedy Algorithm

selection. Figure 5.8 shows the comparison of the random, fixed, and greedy algorithms. Based on the fixed algorithm, we can find the optimal number of servers (e.g., the lowest point in Figure 5.6), and the sizes of the data chunks are the same for each server. For the greedy algorithm, the best server selection set is found. The optimal latency is lower using this algorithm compared with using the fixed algorithm since the sizes of the data chunks are different for the global optimum. However, the premise of the greedy algorithm is that the data can be split into arbitrary sizes, which may not be feasible for many applications in reality, for example, the face recognition algorithm described earlier, since the data representing a face cannot be split. Consequently, a hybrid method that combines the fixed and greedy algorithms (i.e., split the entire data set into different fixed data chunks based on the real conditions, such as different faces of different sizes, then perform greedy algorithms) may be more suitable than the arbitrary-split greedy algorithm for most real applications.

On the other hand, profiling in the cloudlet will incur higher costs (e.g., bandwidth, power, storage, etc.). The random algorithm with redundancy provides a simple method to achieve good latency performance. The random algorithm sends more packets to the network and waits for the earliest response for one task and thus achieves low latency without requiring any profiling of the network. The profiling is reliable when the links/routes are long-lived, while the random algorithm without profiling is more useful in short-lived links/routes or if the mobile, which cannot feasibly perform network profiling, is sending data directly to the cloud without the benefit of a cloudlet.

To sum up, when utilizing a cloudlet in mobile-cloud computing, and performing server selection algorithms (with profiling) or random server selection with redundancy (without profiling), we can obtain significant benefits in terms of response latency.

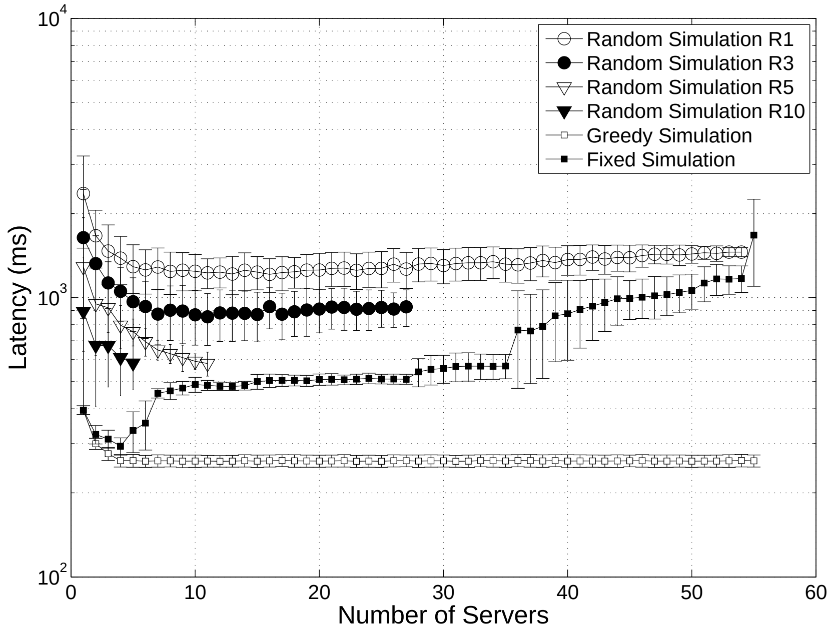

Figure 5.8: Comparison of the random, fixed, and greedy algorithms. Here, data size M=1024 (KB).

6 Conclusions and Future Work

6.1 Conclusions

In this thesis, we show the benefits of utilizing a cloudlet in mobile-cloud computing. With the cloudlet, which can store and update network latency profile information, we can utilize server selection algorithms to bring significant benefits by reducing the overall response latency.

We set up the hardware platform and implemented the software for the MOCHA architecture that utilizes a cloudlet. This setup is later used for network latency measurements. Based on the measured latency data, a profile of the network links/routes is created, including the latency linear model, the latency distribution model, and the latency variance model. Utilizing dynamic profiling, performing the fixed or greedy algorithms for server selection, the response latency is significantly reduced. Moreover, we find that the greedy algorithm is not suitable for non-arbitrary divisible applications (e.g., face recognition) where the hybrid method of fixed and greedy algorithm may be more appropriate. On the other hand, without profiling, which may incur extra costs (e.g., power, storage, and bandwidth), the random algorithm is capable of providing acceptable latency performance with high redundancy. In addition, we argue that the fixed or greedy algorithms, which both highly depend on the profiling, are suitable for long-lived links/routes network environments while the random algorithm is adaptive to short-lived links/routes network environments.

To sum up, utilizing a cloudlet in mobile-cloud computing provides significant benefits from the use of dynamic profiling for optimal server selection.

6.2 Future Work

Profiling on a cloudlet is based on the network latency measurements. Although we perform the measurements for weekdays and weekends and combine them together to generate the profile, it is more reliable to subdivide the profile into different time slots (e.g., daytime or nighttime). Furthermore, a dynamic profile updating strategy should be developed, for example, update the profile when the cloudlet has an otherwise low load and stop profiling when the load in the cloudlet is too high.

Another possible area to look at is the hybrid approach, such as the hybrid method using the fixed and greedy algorithms described in the previous chapter. Moreover, another option worth exploring is adding redundancy to the greedy algorithm. For example, require that 2 copies of all data be sent and use the greedy algorithm to determine how/where to send the data. (e.g., using the greedy algorithm with 2 copies sent). This hybrid method could help with incorrect or invalid profile information, and it provides some level of fault tolerance in the case where a selected server is unavailable.

In this thesis, we mainly focus on reducing the response latency. Nevertheless, cloud resources are not free, especially for advanced services (e.g., larger storage, more computation power, and high bandwidth). Cost is another important concern in mobile-cloud computing. One should take into account both the latency and the costs together to make a balanced selection in achieving an acceptable response latency with a reasonable price. This could be an extension of the work described in this thesis. Furthermore, in addition to on-demand cloud resources (e.g., pay a fixed price for certain resources), there are auction-based cloud resources whose availabilities are controlled by the current bidding prices and the auctions market [11]. In this case, achieving the optimal sever selection will be more challenging and requires further research.

Bibliography

[1] MOCHA, "Mocha research project." http://themochaproject.com/index.html.

[2] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-Vision: Real-time Face Recognition Using a Mobile-Cloudlet-Cloud Acceleration Architecture," *Proc. of IEEE ISCC*, Jul. 2012.

[3] T. Soyata, R. Muraleedharan, J. Langdon, C. Funai, S. Ames, M. Kwon, and W. Heinzelman, "COMBAT: mobile-Cloud-based cOmpute/coMmunications infrastructure for BATtlefield applications," *Proc. of SPIE*, vol. 8403, 84030K-1, 2012.

[4] Wikipedia, "Planetlab." http://en.wikipedia.org/wiki/PlanetLab.

[5] PlanetLab, "Planetlab." http://www.planet-lab.org/.

[6] Wikipedia, "Wireless ad hoc network." http://en.wikipedia.org/wiki/Wireless_ad_hoc_network.

[7] Wikipedia, "Wireless access point." http://en.wikipedia.org/wiki/Wireless_access_point.

[8] Wikipedia, "Windows sockets api." http://en.wikipedia.org/wiki/Winsock.

[9] Wikipedia, "Internet protocol suite." `http://en.wikipedia.org/wiki/TCP/IP`.

[10] V. Paxson, "End-to-end routing behavior in the internet," *Proc.of ACM SIGCOMM*, pp. 41–56, Aug. 1996.

[11] M. Taifi, J. Y. Shi, and A. Khreishah, "Towards auction-based hpc computing in the cloud," *Computer Technology and Application*, vol. 3, pp. 499–509, 2012.