ISLAMIC UNIVERSITY OF TECHNOLOGY

MAC: A framework for on demand offloading mobile application

By:

Rafat-Al-Islam (094420)

Md. Al Maruf (094432)

Supervised by:

Md. Kamrul Hasan, PhD

Assistant Professor

Department of Computer Science and Engineering

Islamic University of Technology

A thesis submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering

Academic Year: 2012-2013

Department of Computer Science and Engineering

Islamic University of Technology.

A Subsidiary Organ of the Organization of Islamic Cooperation.

Dhaka, Bangladesh.

October 2013

Declaration of Authorship

We,
Rafat-Al-Islam (094420)
Md. Al Maruf (094432), declare that this thesis titled,'MAC: A framework for on demand offloading mobile application' and the work presented in it are our own. We confirm that:

- This work was done wholly while in candidature for a Bachelor degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.

Submitted By:

Rafat-Al-Islam (094420)

Md. Al Maruf (094432)

MAC: A framework for on demand offloading mobile application

Approved By:

Prof. Dr. M.A. Mottalib
Head of the Department,
Department of Computer Science and Engineering,
Islamic University of Technology.

Md. Kamrul Hasan, PhD
Thesis Supervisor,
Assistant Professor,
Department of Computer Science Of Technology,
Islamic University of Technology.

ISLAMIC UNIVERSITY OF TECHNOLOGY

Abstract

CSE
Department of Computer and Science Engineering

Bachelor of Science in Computer and Science Engineering

MAC: A framework for on demand offloading mobile application

by
Rafat-Al-Islam (094420)
Md. Al Maruf (094432)

Mobile applications are becoming increasingly ubiquitous and provide ever richer functionality on mobile devices. At the same time, such devices often enjoy strong connectivity with more powerful machines ranging from laptops and desktops to commercial clouds. This paper presents the Architecture and partial implementation of Mobile Application Cloud (MAC), a system that automatically transforms mobile applications to bene?t from the cloud. This System fully offloads the code execution from the mobile device to the device operating in the computational cloud. At runtime the executing part along with other executable file are converted into bytecode and transferred via thread. And later re-integrate the result into mobile device.

Acknowledgements

Special thanks to:

Prof. Dr. M.A. Mottalib
Head of the Department,
Department of Computer Science and Engineering,
Islamic University of Technology

Hasan Mahmud
Assistant Professor
Department of Computer Science and Engineering

All the faculties
System and Software Lab(SSL)

Contents

Declaration of Authorship **i**

Abstract **iii**

Acknowledgements **iv**

List of Figures **vii**

Abbreviations **viii**

1 Introduction **1**
- 1.1 Introduction 1
- 1.2 Motivation 1
- 1.3 Context 2
- 1.4 Related works 2
- 1.5 Problem Summary 3
- 1.6 Solution Approach 3
- 1.7 Layout of the Report 3

2 Background Study **4**
- 2.1 Definition of Cloud Computing 4
- 2.2 VM-based Cloudlets 5
  - 2.2.1 Work Flow of VM-based Cloudlet 5
  - 2.2.2 Benefits and drawback of Cloudlet 6
- 2.3 Clone Cloud 6
  - 2.3.1 Static analysis 6
  - 2.3.2 Dynamic analysis 7
  - 2.3.3 Code partitioning 7
  - 2.3.4 Constraints of Code Partitioning 8
  - 2.3.5 Migration Overview 8
- 2.4 ThinkAir 9
  - 2.4.1 Code Offloading 9
  - 2.4.2 Architecture of ThinkAir framework 10
  - 2.4.3 Constraints of ThinkAir 11

3 Problem Statement 13
3.1 Problem Statement 13

4 Solution Approach 14
4.1 Mobile Application Cloud 14
4.2 Architecture for full application code migration 14
4.3 Work Flow 14

5 Implementation Strategy and Progress 18
5.1 Using Java Netbeans IDE for Establishing Client-Server Connection 18
5.2 JAVA Socket Programming 19
5.3 Advantages and Disadvantages of Java Sockets 19
5.4 Implementation Plan 20
5.5 Implementation 21
5.5.1 Graphical User Interface (Client) 21
5.5.2 Application Handler (Client): 21
5.5.3 Application Proxy (Client) 22
5.5.4 Execution Handler (Server End) 22
5.5.5 Creating the Jar file 23
5.5.6 Execution Engine (Server End) 23

6 Future works and Challenges 25
6.1 Future works and Challenges 25

7 Conclusions 26
7.1 Conclusions 26

Bibliography 27

List of Figures

2.1 Three layers of Cloud Computing 5
2.2 An example of a program, its corresponding static control flow graph, and a partition. 7
2.3 Migration Overview 9
2.4 Overview of the ThinkAir framework locally on the phone 10

4.1 MAC Framework 15

5.1 A socket is an endpoint for communication between two machines . 20
5.2 Graphical User Interface 21
5.3 Application Handler 21
5.4 Snapshot of creating JAR in cmd 23

Abbreviations

API	**A**pplication **P**rogramming **I**nterface
MAC	**M**obile **A**pplication **C**loud
MAUI	**M**obile **A**ssistance **U**sing **I**nterface
SOA	**S**ervice **O**riented **A**rchitecture
SaaS	**S**oftware **as a S**ervice
PaaS	**P**latform **as a S**ervice
IaaS	**I**nfrastructure **as a S**ervice
NDK	**N**ative **D**evelopment **K**it
JVM	**J**ava **V**irtual **M**achine
TCP	**T**ransmission **C**ontrol **P**rotocol
J2SE	**J**ava **S**tandard **E**dition
HTML	**H**yperText **M**arkup **L**anguage

Dedicated to our parents. . . .

Chapter 1

Introduction

1.1 Introduction

Recently the new technology based Mobile devices like smartphones,tablet pcs have gained tremendous popularity for supporting various services of mobile applications (e.g. google apps). As a result wide range of applications are becoming more complex which demands rich resources for computational tasks. For example specch recognition, gaming, face recognition, image processing application mobile devices need rich resources to compute it's tasks locally. Again for rapid increasing of complex applications it is very tough to integrate resources for mobile devices periodically where to run these resource intensive applications on devices we need more computational power,energy,storage and it will also be time consuming task in many cases

1.2 Motivation

As mobile devices can not run these application intesively then mobile cloud computing can benifit us by offloading our applications in cloud. But pre-defined offloading may not be suitable when resources are not available on demand. Now a days we are not much concern about high spped Wi-Fi , bandwidth,wireless technolgies, so on demand full offloading application can overcome partial migration [6] limitation containing complex infrastructure to gain real time user experience result in computation. If we consider about a physical directional game then in that case when we offload the game application after that firstly needs standard and native inputs which should be accurately identified. Secondly for computational environment we need good migration agent to offload the application. These are the challenges we need to consider in offloading purposes.

1.3 Context

In mobile cloud computing, this application offloading has become the main issue. Where proper application offloading and execution attracts to overcome existing difficulties in offloading or migrating application approaces. Seveal research work can be found related to offload application to cloud.

1.4 Related works

For example VM-based cloudlet [1], the cloudlet gives solution providing its existing rich resource server. Mobile devices and cloudlets are connected through WLAN or Wi-Fi. Over here the cloudlet consists of virtual machine that will be run by a private VM overlay delivered by a mobile devices. After the launching the base VM, a request will run the expected or required application on the VM. Now the client guest will be able to use the cloud resources for it's computational task. Here cloudlet provides only VM migration .On demand when the cloudlet doesn't contain the requested application resources then it doesn't provide any offloading or migrating application code to compute the tasks.

CloneCloud [6] is another approach for offloading or partioning the application to cloud. But the same thing happens here where VM migration becomes the main issue again in clonecloud. For method level application computational offloading and smartphone virtualization ThinkAir" [3] is another proposed approach. Thinkair which provides the framework for offloading the application code in cloud. In ThinkAir the main idea behind it is to offload or execute the partial methods of the application in VM based cloud environment. Execution Controller [3] is another vital part to determine which portions of the application will be run locally in mobile devices and which portions or methods should go for execution on cloud. The main challenging part is that to keep track of all the partition of application it creates a huge and complex cloud infrustructure which is not easily maintainable. Partitoning is another concern for application becauses for partitioning we need programmars to modify the codes and where we have to partition for easily attachment at return process. All these complex infrastructure may costly to stand up fully with it. Specially for gaming application it may not be suitable to partially migrating or offloading the application code to cloud.

1.5 Problem Summary

So if the mobile cloud doesn't contain the required resources or applications then the cloud fails to response for the services demanded by clients. In this scenario, migrtaing the unmodified full application code for execution to cloud will be better than partial migtation leaving complex cloud infrastructure.

1.6 Solution Approach

Mobile application cloud (MAC), an application framework which helps to migrate or offload the full demanding code or application code to mobile cloud for execution directly. MAC executes full application successfully and retrieves the desired result on demand.

1.7 Layout of the Report

In this paper, we have proposed for fully migrating or offloading the application code from mobile clients in Mobile Application Cloud (MAC) where we execute the code in Virtual Machine (VM) or emulator.

Chapter 2

Background Study

2.1 Definition of Cloud Computing

Cloud computing is a general term for anything that involves delivering hosted services over the Internet. Cloud computing refers to applications and services that run on a distributed network using virtualized resources and accessed by common Internet protocols and networking standards [9]. "Mobile Cloud Computing at its simplest refers to an infrastructure where both the data storage and the data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and mobile computing to not just smartphone users but a much broader range of mobile subscribers" [10]. Three layers of Cloud Computing: These are the cloud computing services providers categorized among three different layers. The layers are given below: Infrastructure-as-a-Service (IaaS)-the Infrastructure services layer: In the case of IaaS, servers, network devices, and storage disks are made available to organizations as services on a need-to basis. Virtualization (a software technology that uses a physical resource such as a server and divides it up into virtual resources called Virtual Machines-VMs), allows IaaS providers to offer almost unlimited instances of servers to clients, while making cost-effective use of the hosting hardware. Microsoft has been offering IaaS services, either through its own infrastructure or that of its partners. Platform-as-a-Service (PaaS)-the Platform layer: This layer provides a platform for creating applications. PaaS solutions are essentially development platforms for which the development tool itself is hosted in the Cloud and accessed through a browser. With PaaS, developers can build Web applications without installing any tools on their computers and then deploy those applications without any specialized systems administration skills. Software-as-a-Service (SaaS)-the Application layer: This layer includes applications that run off the Cloud and are available to Web users or enterprises on a pay-as-you-go, anytime-anywhere basis. Microsoft's Online Services are an example of SaaS for the enterprise.

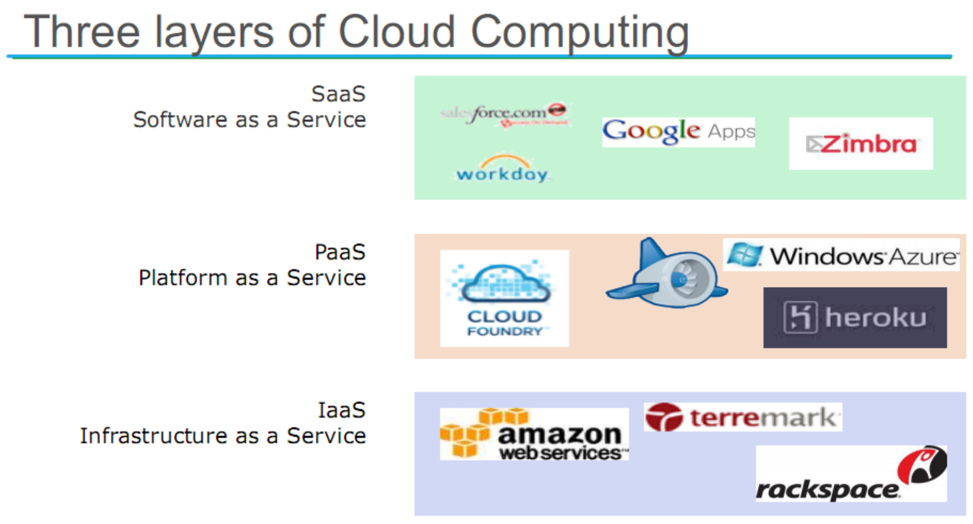


FIGURE 2.1: Three layers of Cloud Computing

In this portion we describe about some of the Existing research works. Most of the works have been lately done using partial code migration. We mainly focus on the working architecture and analyze the techniques of partial code migration.

2.2 VM-based Cloudlets

In this architecture, a mobile user exploits virtual machine (VM) technology to rapidly instantiate customized service software on a nearby cloudlet, and then uses that service over a wireless LAN. The mobile device typically functions as a thin client with respect to the service. A cloudlet is a trusted, resource-rich computer or cluster of computers that is well-connected to the Internet and is available for use by nearby mobile devices.

2.2.1 Work Flow of VM-based Cloudlet

Firstly the mobile device finds the nearby cloudlet and negotiates the use of the cloud. Then sends an private base and a VM overlay. This approach is known as

Dynamic VM synthesis [1]. A small VM overlay is delivered by a mobile device to cloudlet infrastructure that already possess the base VM from which this overlay was derived. The infrastructure applies the overlay to the base to derive the launch VM, which starts execution in the precise state from which the overlay was derived. After the execution VM discards the VM residue.

2.2.2 Benefits and drawback of Cloudlet

Rather than relying on a distant "cloud," the resource poverty of a mobile device can be addressed by using a nearby resource-rich cloudlet. The need for real-time interactive response can be met by low latency, one-hop, high-bandwidth wireless access to the cloudlet. The mobile device functions as a thin client, with all signi?cant computation occurring in the nearby cloudlet. Physical proximity of the cloudlet is essential the end-to-end response time of applications executing in the cloudlet needs to be fast (few milliseconds) and predictable. If no cloudlet is available nearby, the mobile device can gracefully degrade to a fallback mode that involves a distant cloud or, in the worst case, solely its own resources. Full functionality and performance can return later, when a nearby cloudlet is discovered. But the main problem arises when the required resources are not available in the cloudlet.

2.3 Clone Cloud

when we send code partially to the cloud then according to MAUI [Cuervo 2010], the Clone Cloud partitioner automatically identifies costs through static and dynamic code analysis and runs an optimizer to solve partitioning problems, but they went a step further by not asking for the programmer's help (e.g., source annotations).

2.3.1 Static analysis

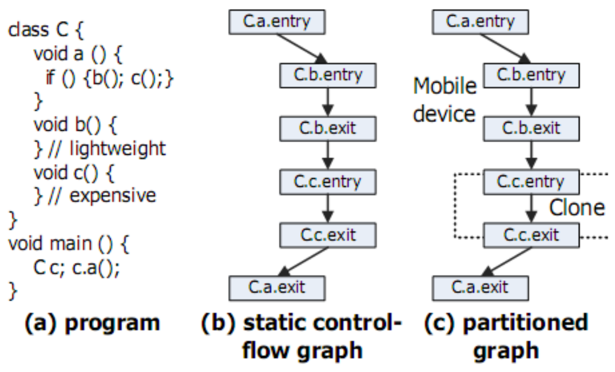
Static analysis looks at the syntactical structure of code and draws conclusions about the program behavior. The partitioner uses static analysis to identify legal choices for placing migration and re-integration points in the code. In principle, these points could be placed anywhere in the code, but we reduce the available choices to make the optimization problem tractable.

2.3.2 Dynamic analysis

Dynamic analysis is not looking at the syntax only, but takes state information into account. In symbolic execution, you are adding assumptions about the possible values of all variables to the statements. The most expensive and powerful method of dynamic analysis is model checking, where you really look at all possible execution states of the system. So performing this requires code analysis tools, but these commercial tools are very expensive.

2.3.3 Code partitioning

The partitioning mechanism in Clone Cloud [6] is offline, and aims to pick which parts of an application's execution to retain on mobile device and which to migrate



```
class C {
    void a () {
      if () {b(); c();}
    }
    void b() {
    } // lightweight
    void c() {
    } // expensive
}
void main () {
    C c; c.a();
}
```

FIGURE 2.2: An example of a program, its corresponding static control flow graph, and a partition.

to the cloud. So any application executed in the cloud that search for an application VM after the code analysis can be partitioned. An approach MAUI project [Cuervo 2010], the programmer is not bound to write the code in special form or annotate it in non-standard way and the source code is also not required. Partition of a code is output of partitioning mechanism where the code is divided into some execution point from where the code can be migrated to an application VM. The applications code will be migrated according to created execution points between the devices and the clone.

2.3.4 Constraints of Code Partitioning

There are mainly three general constraints faced partitioning codes. The constraints are as follows:

I. Methods that access specific features of a machine must be pinned to the machine.

Explanation: If a method uses a local resource such as the location service (e.g., GPS) or sensor inputs (e.g., microphones) in a mobile device, the method must be executed on the mobile device. This primarily concerns native methods, but also the main method of a program. The analysis marks the declaration of such methods with a special annotation M-for Mobile device. We manually identify such methods in the VM's API (e.g., VM API methods explicitly referring to the camera); this is done once for a given platform and is not repeated for each application. We also always mark the main method of a program. We refer to methods marked with M as the VM method set.

II. Methods that share native state must be collocated at the same machine.

Explanation: An application may have native methods that create and access state below the VM. Native methods may share native state. Such methods must be collocated at the same machine as our migration component does not migrate native state For example, when an image processing class has initialize, detect, fetch result methods that access native state, they need to be collocated at the same machine. To avoid a manual-annotation burden, native state annotations are inferred automatically by the following simple approximation, which works well in practice. We assign a unique annotation NatC to all native methods declared in the same class C; the set VNatC contains all methods with that annotation.

III. Prevent nested migration

Explanation: With one phone and one clone, this implies that there should be no nested suspends and no nested resumes. Once a program is suspended for migration at the entry point of a method, the program should not be suspended again without a resume, i.e., migration and re-integration points must be defined.

2.3.5 Migration Overview

Migration in CloneCloud is done using after deciding the process to migrate suspend the code at that position and then migrate the code along with the static

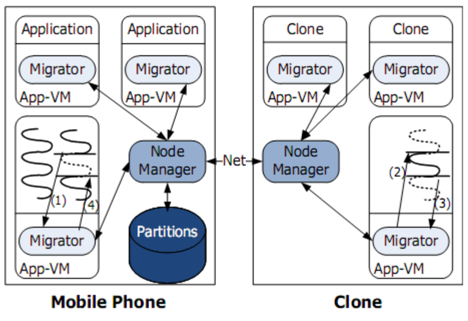


FIGURE 2.3: Migration Overview

inputs. Then after executing the code in the CloneCloud it sends back the result to mobile phone. This process is known as Suspending and Resuming.

2.4 ThinkAir

ThinkAir [2] is another research work based on partial code migration. ThinkAir is a framework that makes it simple for developers to migrate their smartphone applications to the cloud. ThinkAir exploits the concept of smartphone virtualization in the cloud and provides method-level computation offloading. ThinkAir provides an ef?cient way to perform on-demand resource allocation, and exploits parallelism by dynamically creating, resuming, and destroying VMs in the cloud when needed. ThinkAir addresses these two aspects in mobile clouds. Supporting on-demand resource allocation is critical as mobile users request different computational power based on their workloads and deadlines for tasks, and hence the cloud provider has to dynamically adjust and allocate its resources to satisfy customer expectations

2.4.1 Code Offloading

MAUI [Cuervo 2010], a system that enables fine-grained energy-aware of?oad of mobile code to the infrastructure. Previous approaches to these problems either relied heavily on programmer support to partition an application, or they were coarse-grained requiring full process (or full VM) migration. MAUI uses the benefits of a managed code environment to offer the best of both worlds: it supports fine-grained code of?oad to maximize energy savings with minimal burden on the programmer. MAUI decides at run-time which methods should be remotely executed, driven by an optimization engine that achieves the best energy savings possible under the mobile device's current connectivity constrains. In our evaluation, we show that MAUI enables: 1) a resource-intensive face recognition application that consumes an order of magnitude less energy, 2) a latency-sensitive arcade game application that doubles its refresh rate, and 3) a voice-based language translation application that bypasses the limitations of the smartphone environment by executing unsupported components remotely

2.4.2 Architecture of ThinkAir framework

I. Programmer API:

Since the execution environment is accessed indirectly by the developer, ThinkAir provides a simple library that, coupled with the compiler support, makes the programmer's job very straightforward, any method to be considered for offloading is annotated with @Remote.

II. Compiler:

A key part of the ThinkAir framework, the compiler comes in two parts: the Remoteable Code Generator and the Customized Native Development Kit (NDK). The Remoteable Code Generator is a tool that translates the annotated code.

III. Execution Controller:

The Execution Controller drives the execution of remote able methods. It decides whether to of?oad execution of a particular method, or to allow it to continue

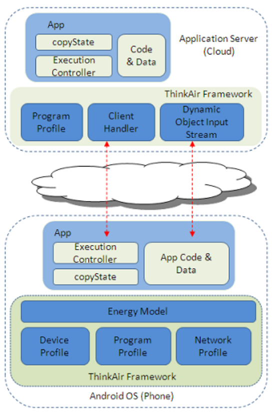


FIGURE 2.4: Overview of the ThinkAir framework locally on the phone

The decision depends on data collected about the current environment as well as that learnt from past executions. When a method is encountered for the first time, it is unknown to the Execution Controller and so the decision is based only on environmental parameters such as network quality: for example, if the connection is of type WiFi, and the quality of connectivity is good, the controller is likely to offload the method. At the same time, the profilers start collecting data. On a low quality connection, however, the method is likely to be executed locally. If and when the method is encountered subsequently, the decision on where to execute it is based on the methods past invocations, i.e., previous execution time and energy consumed in different scenarios, as well as the current environmental parameters.

IV. Execution flow:

On the phone, the Execution Controller first starts the profilers to provide data for future invocations. It then decides whether this invocation of the method should be offloaded or not. If not, then the execution continues normally on the phone.

APPLICATION SERVER:

I. Client Handler:

The Client Handler executes the ThinkAir communication protocol, managing connections from clients, receiving and executing of?oaded code, and returning results. To manage client connections, the Client Handler registers when a new application, i.e., a new instance of the ThinkAir Execution Controller, connects. If the client application is unknown to the application server, the Client Handler retrieves the application from the client, and loads any required class de?nitions and native libraries. It also responds to application- level ping messages sent by the Execution Controller to measure connection latency. Following the initial connection set up, the server waits to receive execution requests from the client.

2.4.3 Constraints of ThinkAir

I. To keep track of all the partitions of application this makes a huge and complex cloud infrastructure.

II. Programmers needed to modify the codes. The codes that are decided for offloading must have human annotation based structure. This increases the complexity and programmer himself must define the annotation. For this reason the programmer is needed to skilled. This process of defining is time consuming.

III. Where we have to partition for easily attachment of output at return process.

Chapter 3

Problem Statement

3.1 Problem Statement

Now if we summarize the problem what if the resources demanded by the device are not available in the cloud. Now for partial code offloading it requires to keep track of all the states of the code. The good programmers are needed to manually modify the codes for partitioning purposes. For making this dynamic we need static analyzer and dynamic analyzer which leads to some constraints discussed in our previous section 2.2.4 which may hamper our desired output. All these actually can lead us to complex and time consuming computational tasks. This leads to our proposed approach trying to eradicate these drawbacks.

Chapter 4

Solution Approach

4.1 Mobile Application Cloud

According to the problem statement instead of offloading the code manually we propose to send the full application code to the cloud infrastructure. Thus by sending full application code to the cloud there is no need of dynamic and static analyzer. Because we don't need to keep track of the code states if send the full code to the cloud. Again by full offloading the code removes the general's constraints of native input and nested code in the application. We propose a framework Mobile Application Cloud (MAC) which will provide full application offloading. Mobile Application Cloud (MAC) is an application framework which helps to migrate or offload the full demanding code or application code to mobile cloud for execution directly. MAC executes full application successfully and retrieves the desired result on demand. In the later portion we also described about an architecture which supports our approach.

4.2 Architecture for full application code migration

This is our proposed architecture. We named it Mobile Application Cloud (MAC)

4.3 Work Flow

Keyboard Input

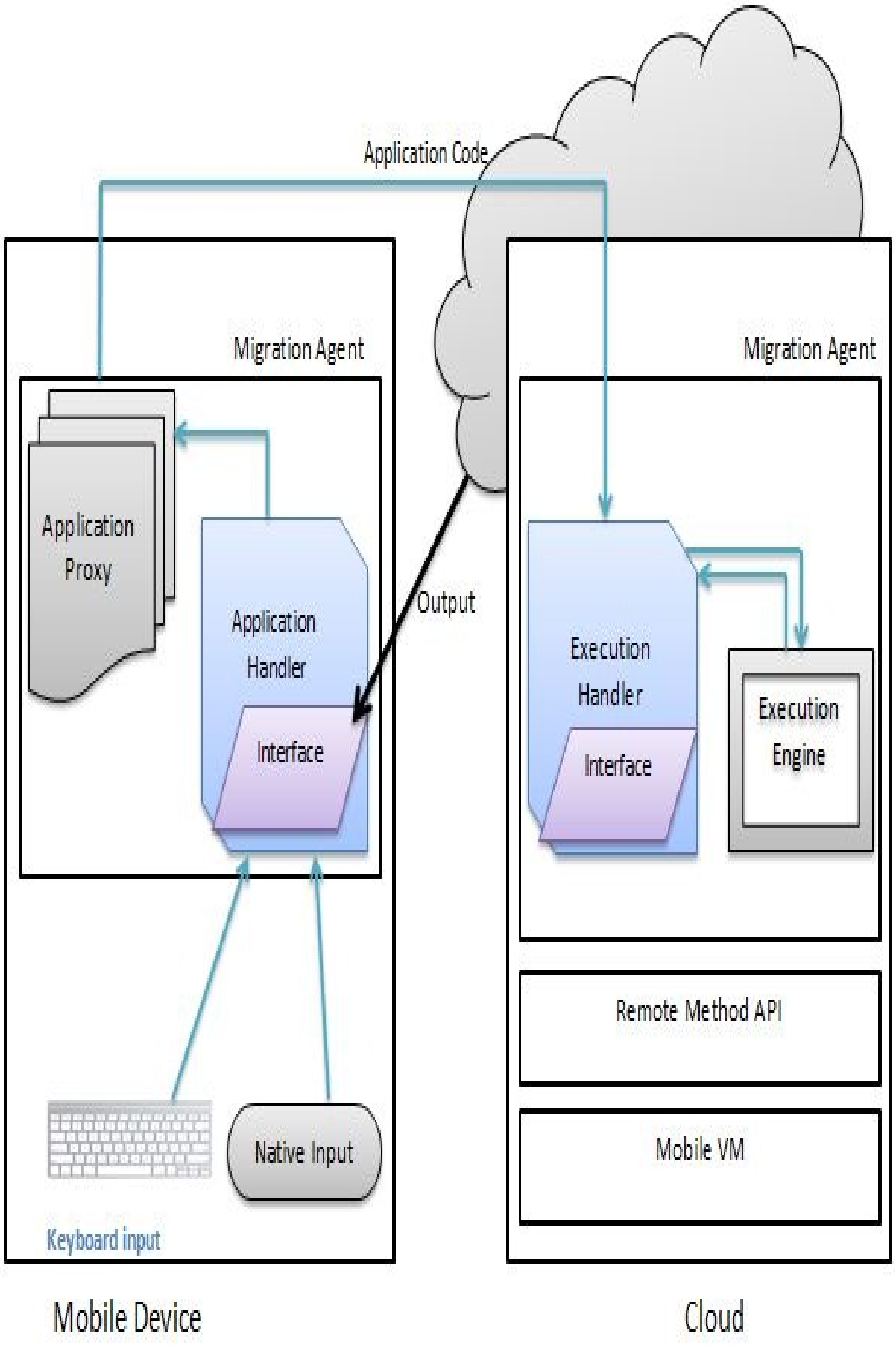


Figure 4.1: MAC Framework

This is termed as the standard input. The user of the mobile device gives the input using keyboard. For this any standard mobile keyboard is do the job.

Native input

For example if we consider a game that needs continuous GPS input of the gamer position and decide on its input. As we are sending this input to another machine for processing so we need data distribution which can be achieved through native input module. For designing this module we can use the existing input output libraries available in java(J2SE) or google android platform.

Migration Agent

Migration Agent handles the migration techniques. Migration techniques are required to dynamically move computation between mobile nodes and cloud nodes. It has low overhead especially when using in mobile devices where processing power and resources are very limited. For our purpose we will use Application?level task migration. This can be achieved through rewriting bytecode using javaSplit, Rewriting source code using javaGo. Asynchronous exception we don't need to add migration point and state-polling codes. In our case migration agent(Mobile end) send the entire application to cloud and also send real time input from the mobile devices to the cloud using pro-active migration(triggered by the program itself) which has Migration manager would then receive the request, choose the appropriate destination and send the data. Migration agent (Cloud end) after receiving the data will execute the code and send its output to the interface portion of the migration agent(Mobile End).

Migration Agent (Mobile End)

I. Application Handler:

Application handler is a part of the migration agent that handles managing connections from clients, receiving and executing of?oaded application code. To manage client connections, the Client Handler registers when a new application, i.e., a new instance of the Mobile Application Cloud (MAC) execution Controller, connects. If the client application is unknown to the application server, the Client Handler retrieves the application from the client, and loads any required class de?nitions and native libraries. It also responds to application-level ping messages sent by the Execution Controller to measure connection latency and returning results. Following the initial connection set up, the server waits to receive execution requests from the client. A request consists of necessary data: containing object, requested method, parameter types, parameters themselves, and a possible request for extra computational power. If there is no request for more computational power, the Application Handler proceeds much as the client would, the remote able method is called using Java re?ection and the result, or exception if thrown, is sent back. Client Handler distributes the task in cloud, collects and sends results back to the client. Along with the return value, the Client Handler also sends pro?ling data for future of?oading decisions made by the Execution Controller at the client side.

II. Application Proxy:

Proxies (often called intermediaries in the SOA world) are hardware or software solutions that sit between the client and the server and do something to requests and sometimes responses. Application discovers the potential network determination and also identifies the amount of bandwidth required in performing the full application migration.

III. Interface

In the interface the result back from the cloud is shown. This can be widget in the mobile device for the visual purpose. Interface is controlled by the Migration Agent of the Mobile end.

Migration Agent (Cloud end):

Execution Handler:

Execution handler catches the migrated code, then captures the migrated state and introduces the migrated application to Execution engine.

Execution Engine:

After receiving the application from the Execution handler it selects a application type to execute. Selection can be automatic. After executing the task it sends back the result to Execution handler to transfer the result to the migration agent of mobile end.

Remote Method API:

It will work along with the mobile VM to support the execution of the transferred code in the cloud. The standard libraries are used for instances.

Chapter 5

Implementation Strategy and Progress

5.1 Using Java Netbeans IDE for Establishing Client-Server Connection

Now if we summarize the problem what if the resources demanded by the device are not available in the cloud. Now for partial code offloading it requires to keep track of all the states of the code. The good programmers are needed to manually modify the codes for partitioning purposes. For making this dynamic we need static analyzer and dynamic analyzer which leads to some constraints discussed in our previous section 2.2.4 which may hamper our desired output. All these actually can lead us to complex and time consuming computational tasks. This leads to our proposed approach trying to eradicate these drawbacks.

Java is a general-purpose, concurrent, class-based, object-oriented computer programming language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers ”write once, run anywhere” (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java applications are typically compiled to bytecode (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture. Java is, as of 2012, one of the most popular programming languages in use, particularly for client-server web applications, with a reported 10 million users.

There were five primary goals achieved using Java language: 1. simple, object-oriented and familiar 2. robust and secure 3. architecture-neutral and portable 4. high performance 5. interpreted, threaded, and dynamic

5.2 JAVA Socket Programming

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server. When the connection is made, the server creates a socket object on its end of the communication. The client and server can now communicate by writing to and reading from the socket. The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them. The following steps occur when establishing a TCP connection between two computers using sockets: The server instantiates a ServerSocket object, denoting which port number communication is to occur on. 1. The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.

2. After the server is waiting, a client instantiates a Socket object, specifying the server name and port number to connect to.

3. The constructor of the Socket class attempts to connect the client to the specified server and port number. If communication is established, the client now has a Socket object capable of communicating with the server.

4. On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.

5.3 Advantages and Disadvantages of Java Sockets

Advantages of Java Sockets:

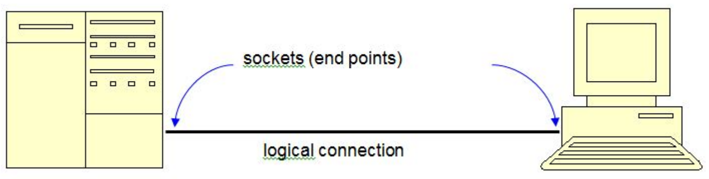


Figure 5.1: A socket is an endpoint for communication between two machines

Sockets are flexible and sufficient. Efficient socket based programming can be easily implemented for general communications. Sockets cause low network traffic. Unlike HTML forms and CGI scripts that generate and transfer whole web pages for each new request, Java applets can send only necessary updated information.

Disadvantages of Java Sockets:

Security restrictions are sometimes overbearing because a Java applet running in a Web browser is only able to establish connections to the machine where it came from, and to nowhere else on the network despite all of the useful and helpful Java features, Socket based communications allows only to send packets of raw data between applications. Both the client-side and server-side have to provide mechanisms to make the data useful in any way.

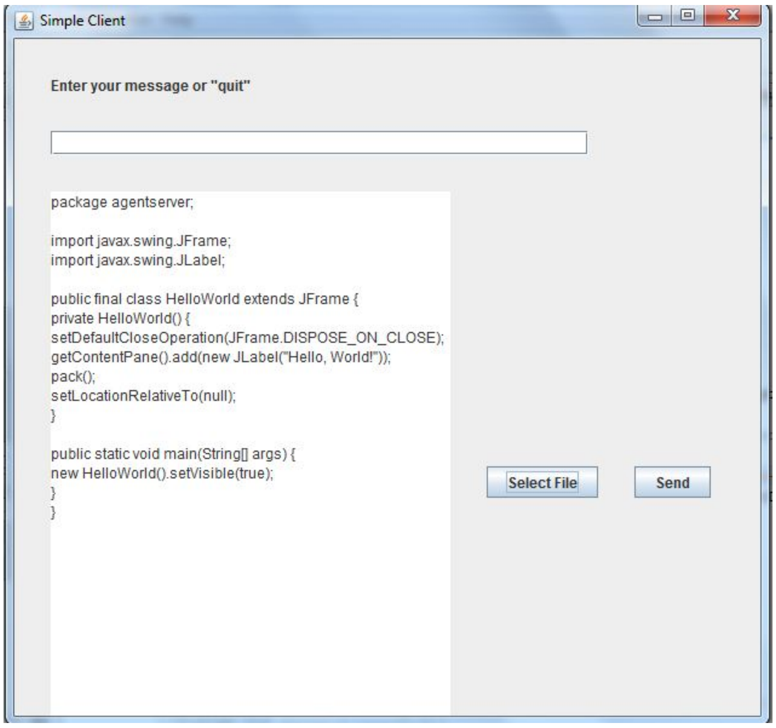
5.4 Implementation Plan

After deciding on the choosing the Java as programming language then Socket programming became a better approach on implementing our Architecture. We will use socket programming for establishing connection between the Client and Server. After establishing the connection we send the application class files into the server through our Graphical user Interface of the Client Part. Then read the input file in the form bytecode by an InputStream. After the receiving the bytecode form the client, the code is rewritten in the server. Then the rewritten code is converted to .jar file. The code the executed and result is stored. Finally the result is then sent to the client.

5.5 Implementation

5.5.1 Graphical User Interface (Client)

The client after launching the application the user has the freedom to choose any file. Then by the send button user can send the data to server.



```
package agentserver;

import javax.swing.JFrame;
import javax.swing.JLabel;

public final class HelloWorld extends JFrame {
private HelloWorld() {
setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
getContentPane().add(new JLabel("Hello, World!"));
pack();
setLocationRelativeTo(null);
}

public static void main(String[] args) {
new HelloWorld().setVisible(true);
}
}
```

FIGURE 5.2: Graphical User Interface

5.5.2 Application Handler (Client):

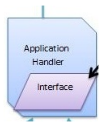


FIGURE 5.3: Application Handler

The code for handling the input is given below:

```
dis = new DataInputStream (new FileInputStream (path));
datainBytes = new byte[dis.available()];
dis.readFully(datainBytes);
dis.close();
content = new String(datainBytes, 0, datainBytes.length);
```

Here for processing the input from the client a DataInputStream is created and input file read as a stream and send the stream to server.

5.5.3 Application Proxy (Client)

Application discovers the network and also identifies the amount of bandwidth required in performing the full application migration. The pseudo code for implementing this feature is given below:

```
    Socket client = null;
    String serverAddr = "localhost";
    int serverPort = 8888;
    PrintWriter out
   client = new Socket(serverAddr, serverPort);
   System.out.println("Client: " + client);
   out = new PrintWriter(client.getOutputStream());
```

In our implementation we have a fixed server address and server port which is localhost and 8888 respectively. Then using the server address and server port the application proxy creates connection between the server.

5.5.4 Execution Handler (Server End)

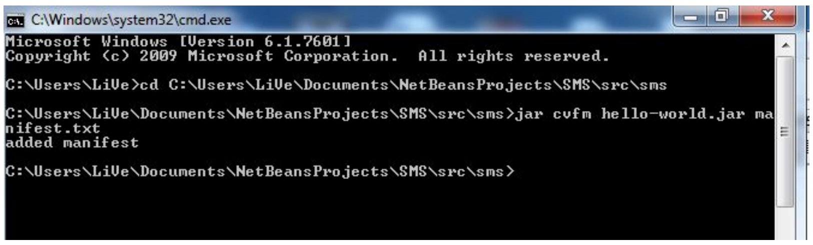
After application handler has sent the code to server, Execution handler in the server end receives the code in bytecode form. Then the received information is again rewritten in the server byte by byte. The code for this feature is given below:

```
   try {

          String Client_id=socket.getInetAddress().getHostAddress().toString();
           String mid;
           System.out.println("You are welcome");
           br=newBufferedReader(new InputStreamReader(socket.getInputStream()));
            int byteRead;
        // Block until the client closes the connection, results in read() returns -1
            while ((byteRead = br.read()) != -1) {

                //prints the message
                message = (char) byteRead + br.readLine().toString();
               System.out.println("message: " + message);
//Rewriting the code in the server
          try
             {
               FileWriter fstream = new FileWriter("hello-world.java", true);
 //true tells to append data.
                BufferedWriter out = new BufferedWriter(fstream);
               out.write(message );
                out.newLine();
                out.close();
```

5.5.5 Creating the Jar file

The rewritten file is converted to .jar file by creating manifest.txt including the required classes. Then by writing the following code in the windows cmd screen .Jar file of that file is created.

Java cvfm name.jar manifest.txt [/src/*.class][optional]



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\LiVe>cd C:\Users\LiVe\Documents\NetBeansProjects\SMS\src\sms

C:\Users\LiVe\Documents\NetBeansProjects\SMS\src\sms>jar cvfm hello-world.jar ma
nifest.txt
added manifest

C:\Users\LiVe\Documents\NetBeansProjects\SMS\src\sms>
```

FIGURE 5.4: Snapshot of creating JAR in cmd

5.5.6 Execution Engine (Server End)

In this portion of our Implementation we executed the jar file that is created by the Execution handler. This jar file is executed.

The jar file created by creating a new process by the server and starting a buffered output stream for capturing the output. The code is given below:

```
   Process p = Runtime.getRuntime().exec("cmd /c start /temp/hello-world.jar");


BufferedInputStream bis=new BufferedInputStream(p.getInputStream());
        synchronized (p) {
            p.waitFor();
        }
       System.out.println(p.exitValue());
        int b=0;
        while((b=bis.read()) >0)
            System.out.print((char)b);
      }
```

Then by capturing the output it is redirected towards the client.

Chapter 6

Future works and Challenges

6.1 Future works and Challenges

The main challenges we faced are there are no programming language still specified to implement any cloud. There is no java API for cloud computing. Input and Output redirection is a challenging process. The codes or Algorithms are not available for the previous cloud implementation. In our future works we intend to implement our code for in Android. We will try integrating the features as demanded by our architecture.

Chapter 7

Conclusions

7.1 Conclusions

Mobile cloud computing is one of mobile technology trends in the future since it combines the advantages of both mobile computing and cloud computing, thereby providing optimal services for mobile users. We present Mobile Application Cloud (MAC), a framework for of?oading mobile computation to the cloud. We are continuing the development of several key components of MAC. Using this architecture we can remove the current limitations of mobile cloud computing. We are hoping full that full offloading will give better performance than the existing approaches.

Bibliography

[1] M. Satyanarayanan, P. Bahl, R. C aceres and N. Davies, “The Case for VM-Based Cloudlets in Mobile Computing,” *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[2] S. Kosta, A. Aucinas, P. Hui, R. Mortier and X. Zhang, “Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code ofoading,” *in Proc. of IEEE INFOCOM*, 2012.

[3] Muhammad Shiraz and Abdullah Gani, “Mobile Cloud Computing: Critical Analysis of Application Deployment in Virtual Machines,” *In ICICN* , 2012.

[4] Hoang T. Dinh, Chonho Lee, Dusit Niyato,and Ping Wang, “A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches , 2011.

[5] Steven Osman, Dinesh Subhraveti, Gong Su, and Jason, “The Design and Implementation of Zap: A System for Migrating Computing Environments,” *Appears in Proceedings of the 5th Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, December 2002.

[6] B.G. Chun, S. Ihm, P. Maniatis, M. Naik and A. Patti, “Clone Cloud: Elastic Execution between Mobile Device and Cloud,” *EuroSys11 Salzburg Austria ACM Press*, April 1013, 2011.

[7] Sarah Perez, “Why Cloud Computing is the Future of Mobile, August 4th, 2009.

[8] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Chandra and P. Bahl, “Maui: Making smartphones last longerwith code ofoad,” *in Proc. of MobiSys*, 2010.

[9] [Online]. http://www.slideshare.net/aidy.allidm/cloud-introduction-12706414

[10] [Online]. http://www.mobilecloudcomputingforum.com/

[11] [Online]. http://www.microsoft.com/india/msindia/perspective/interfaces_cloud_three_layers.aspx