Cloud Computing Frameworks for Food Recognition from Images

by

Sri Vijay Bharat Peddi

Thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of the requirements

For the degree of Masters of Applied Science in

Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

School of Electrical Engineering and Computer Science

University of Ottawa

Abstract

Distributed cloud computing, when integrated with smartphone capabilities, contribute to building an efficient multimedia e-health application for mobile devices. Unfortunately, mobile devices alone do not possess the ability to run complex machine learning algorithms, which require large amounts of graphic processing and computational power. Therefore, offloading the computationally intensive part to the cloud, reduces the overhead on the mobile device. In this thesis, we introduce two such distributed cloud computing models, which implement machine learning algorithms in the cloud in parallel, thereby achieving higher accuracy. The first model is based on MapReduce SVM, wherein, through the use of Hadoop, the system distributes the data and processes it across resizable Amazon EC2 instances. Hadoop uses a distributed processing architecture called MapReduce, in which a task is mapped to a set of servers for processing and the results are then reduced back to a single set. In the second method, we implement cloud virtualization, wherein we are able to run our mobile application in the cloud using an Android x86 image. We describe a cloud-based virtualization mechanism for multimedia-assisted mobile food recognition, which allow users to control their virtual smartphone operations through a dedicated client application installed on their smartphone. The application continues to be processed on the virtual mobile image even if the user is disconnected for some reason. Using these two distributed cloud computing models, we were able to achieve higher accuracy and reduced timings for the overall execution of machine learning algorithms and calorie measurement methodologies, when implemented on the mobile device.

Acknowledgements

I am very grateful to Prof. Shervin Shirmohammadi, who gave me the incredible opportunity of being part of a research project that has excited and inspired me from day one.

He has also offered me research assistantship to support my studies, provided all the equipment necessary for my experiments and directed me towards the right path to make the right choices in my research and career.

I would like to take this opportunity to also thank all of my teammates, including Dr. Abdulsalam Yassine, Pallavi Kuhad, and Parisa, who have been vital in the success of my thesis. I am very fortunate to be part of this enthusiastic team, with which I always enjoy working. We have achieved some amazing heights together. I am also very appreciative of the kind support from Dr. Abdulsalam Yassine, who always shared some great insights into our project and made it successful. I have learned a lot in the company of Parisa, who has always been supportive and shared knowledge from her experiences. I would also like to thank Pallavi, who was also completing her Master's along with me and has been an incredible person to work with. She was always full of novel ideas and a passion that drove her to bring some unique perspectives to the table.

My professor and team have always been kind in nurturing my ideas, ultimately allowing them to turn into ones that were viable for implementation.

I would also like to thank all of the members of the Discover Lab. Being a part of this lab gave me this unique opportunity to witness some interesting projects, which always motivated me in my own work.

I would like thank my parents, who have always been caring and supportive in my decision to pursue a Master's degree in Canada. My Dad, who is a scientific researcher himself, always contributed novel ideas and encouraged me to think differently. I'd also like to thank my younger brother Sharat, who always enjoyed talking to me about my thesis and who shared some out-of-the-box opinions that encouraged me to pursue my thesis.

Table of Content

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

High calorie intake has a severe impact on people's health. Studies show that a number of diseases are a result of excessive calorie intake by humans. High calorie values in food that is nutritionally poor leads to systemic inflammation, reduced insulin sensitivity, and clusters of metabolic abnormalities, including obesity, hypertension, dyslipidemia, and glucose intolerance [1]. From these facts, we can conclude that excessive calories, when consumed through food, are harmful to the human body and need to be kept in check.

One way to keep a check on calorie intake is by using applications installed in smartphones, which can help users monitor their calorie consumption. Smartphones have become an integral part of people's daily lives. People are dependent on smartphones for communication, shopping via mobile websites (m-commerce), and even for maintaining their health and fitness. With advancements in smartphone's hardware capabilities, a number of new and improved sensors have been built into smartphones, further empowering developers to create applications that make efficient use of these sensor capabilities. Specifically the health and fitness applications have benefited significantly from these sensors. Fitness tracker applications on phones allow us to keep track of

calories burned by calculating the number of steps we have climbed, the number of miles we have run or biked, etc. However, there is still a need for applications that can assist us through a proactive approach of checking what we eat, how much to eat, and calorie intake. To run such an application, a mobile phone should have the capacity to implement complex machine learning algorithms and calorie measurement techniques.

Although advancements in smartphone hardware have equipped them to handle parallel and complex processing, there is still the problem of the physical resources of mobile devices being limited, to some extent, when applications demand intensive computations. Cloud services, address these issues by offloading a portion of the processing to the cloud and by making use of cloud resources to efficiently run such complex algorithms.

1.2 Problem Statement

One of the main problems we address is the limited ability of mobile phones to run complex, computing-intensive algorithms, like deep learning, calorie measurement applications, etc. Unfortunately, mobile devices alone do not possess the ability to run complex machine learning algorithms that require high amounts of graphic processing and computational power.

Another problem is that offloading content or processing to the cloud could impact the response times to users' interactions with the mobile application. For example, it would take a mobile phone less time to switch from one mobile activity to another if everything

were installed on the phone. Moreover, offloading content from a mobile application to the cloud involves several dependencies, such as the network quality, the bandwidth, the cloud vendor providing the services, the time it takes to start the cloud instances, the cost to maintain the cloud instances, and the requirements of the cloud administrator (in some scenarios). Hence, a standalone application would be faster in terms of response time to the user, since it wouldn't be dependent on the above-mentioned cloud resources. However, it would be devoid of the computing power needed to run complex algorithms, such as in our scenario, which involves a requirement for complex algorithms that the mobile phone alone could not run.

During our implementation of the MapReduce model (which will be explained in further sections), we faced several issues. Firstly, the initial time needed to distribute the data and process it across resizable Amazon EC2 instances was considerably higher for smaller training sets of data. Secondly, the client-server architecture model required us to re-write the computer vision code to allow it to run in a cloud-based MapReduce environment.

While implementing our other cloud-based virtualization model, we faced major challenges in designing the virtualization architecture, which was otherwise easy to design on local server. During its design phase, we had to make use of virtualization extensions, and then create virtual images on the top of these extensions to give them an additional performance boost. Since we had to depend on cloud-based services from Amazon, we were limited by their services in terms of designing our virtualization architectures.

1.3 Thesis Goals

The goal of this work is to implement a distributed cloud computing model that will enable our smartphone application (Eat Healthy Stay Healthy, or EHSH) to offload a major part of its intensive computing to the cloud. We would like to explore two cloud computing models that could be integrated with existing smartphone capabilities to run our mobile application (EHSH). This would allow us to intelligently offload intensive computations from the smartphone to the cloud in a manner would make efficient use of both of systems' capabilities. We also take into account the scalability of the application by further introducing the concepts of parallel computing and virtualization mechanism in the cloud. We will investigate some of the issues that arise in the parallel computing model and how the cloud virtualization mechanism addresses them without affecting the overall functionality of the mobile application. In designing this model, we have taken into account its ability to run autonomously, without any user intervention (e.g., from the user using the phone or from the administrator), as well as its ability to make independent decisions (using a decision algorithm) regarding how to perform the computing.

1.4 Thesis Contributions

- We propose a parallel Support Vector Machine (SVM) training mechanism in a cloud computing environment, using the MapReduce technique to facilitate distributed machine learning used for recognizing food objects.
- We describe a cloud-based virtualization mechanism for a multimedia-assisted mobile food recognition application. Our mechanism allows users to control their virtual smartphone operations through a dedicated client application installed on the smartphone, while the processing of the application continues to run on the virtual mobile image—even if the user is disconnected for some reason. With our mechanism, the mobile environment is emulated on the cloud in such a manner that users always feel as if the application is being run on the smartphone (while, in reality, it is running virtually on the remote server in the cloud), hence overcoming the limited capability of smartphones to run intensive machine learning.
- We propose an approach in which the mobile session of our mobile application, EHSH, be get swapped to a cloud session running the same mobile application. This will enable faster processing times for running the machine learning algorithm (in our case, deep learning) and for measuring calories with limited resource usage.
- We also propose a decision-making algorithm that will enable the cloud server to choose among various cloud servers for the task of image processing. Our tests show that the virtual swap, along with the decision-making algorithm, is able to provide the necessary computational power and improve the calorie measurement processing timing by 20.5% for the smartphone.

1.5 Research Publications

- Parisa Pouladzadeh, Pallavi Kuhad, Sri Vijay Bharat Peddi, Abdulsalam Yassine, and Shervin Shirmohammadi, "Mobile Cloud Based Food Calorie Measurement" in the 4th International IEEE Workshop on Multimedia Services and Technologies for E-health (MUST-EH 2014), July 14, 2014 Chengdu, China, in conjunction with IEEE ICME 2014.
- Parisa Pouladzadeh, Sri Vijay Bharat Peddi, Pallavi Kuhad, and Shervin Shirmohammadi, "A MapReduce Parallel Classifier for Cloud Based Food Recognition" in the International Conference on Next Generation Computing and Communication Technologies [ICNGCCT]-2014, Dubai, pg. 142-147.
- Sri Vijay Bharat Peddi, Abdulsalam Yassine, Shervin Shirmohammadi, "Cloud based Virtualization for a Calorie Measurement e-Health Mobile Application" in the 4th International IEEE Workshop on Multimedia Services and Technologies for E-health (MUST-EH 2015), June 29-July 3, 2015, Torino, Italy, in conjunction with IEEE ICME 2015.

Journal Publications

- Parisa Pouladzadeh, Sri Vijay Bharat Peddi, Pallavi Kuhad, and Shervin Shirmohammadi, "A Virtualization Mechanism for Real-Time Multimedia-Assisted Mobile Food Recognition Application in Cloud Computing" in *Cluster Computing Journal 2015*.

1.6 Thesis Organization

This thesis is organized as follows. In Chapter 2, we discuss the background and related work, in which the related work focuses on the cloud computing models discussed in this work and their implementation in health care applications. We also discuss in detail the basic architecture of the two cloud computing models proposed in this thesis. In Chapter 3, we introduce the MapReduce cloud model used to implement the parallel SVM classifier. We also elaborate the cloud virtualization model and discuss the components that serve as part of the cloud model. In Chapter 4, we discuss the implementation of these cloud computing models in detail and present the decision algorithm implemented in the cloud model. In Chapter 5, we elaborate on the computed results with respect to each of the mentioned models. In Chapter 6, we summarize the work and discuss in detail about how the proposed models have contributed to an enhanced user experience of the EHSH smartphone application in terms of time, cost, resource consumption, and accuracy. We also discuss possibilities for future work.

Chapter 2

Background

2.1 Distributed Cloud Computing Model

To run complex algorithms as part of the smartphone application, we need to make sure it is implemented as a distributed application, wherein part of the application runs using external resources. As described by [2], we can differentiate between distributed applications and standalone applications, which only require the resources available on the smartphone to execute. In distributed cloud computing, part of the application runs on the phone, and another part executes on one or more external machines. Moreover, when we refer to the term distributed cloud computing, we discuss, not only how the resource-intensive task will be processed in the cloud in parallel, but also how users will connect to the application simultaneously.

Because the major part of the intensive computation is performed on the cloud server, the user will never realize the heavy computing algorithm that runs in the background to achieve food object recognition. This factor hugely benefits the user, since it results in a negligible amount of battery consumption and a lower processing power required from the user's mobile device, since the entirety of the processing is offloaded to cloud. To implement the mobile application as a distributed application, we propose two

approaches, both of which use the capability of the cloud and the mobile smartphone. The first approach is the MapReduce model (as explained in Section 2.1.1), wherein the machine learning algorithm (SVM) is run in parallel on various cloud instances. The second one is the Cloud Virtualization methodology, wherein, instead of offloading the computation to the cloud, the system virtually swaps from the mobile session to a cloud session and then back to the mobile session (when it needs to access the mobile device's sensor). This provides the system the flexibility to choose its session based on its requirements. The next section (2.1.1) explains the MapReduce model in detail.

2.1.1 MapReduce Model

Cloud computing has been widely recognized as the next-generation computing infrastructure. Cloud computing offers advantages through allowing users to access infrastructure (e.g., servers, networks, and storage) and software (e.g., application programs) provided by cloud providers (e.g., Google, Amazon) at a low cost. Mobile cloud computing integrates cloud computing into the mobile environment, overcomes obstacles related to performance and environment, and improves communication across different information formats [1]. Cloud-based systems also allow health professionals and patients to access medical reports wherever they are. This thesis shows the advantages of using the cloud computing paradigm in health care problems—namely, food recognition and calorie measurements. We propose a system for food recognition that uses the SVM training mechanism in a cloud computing environment with the MapReduce technique for distributed machine learning [4] [5][6].

2.1.1.1 Hadoop and MapReduce

Hadoop is an open source Java implementation of Google's MapReduce algorithm, along with an infrastructure to support distribution over multiple machines. This includes its own file system, Hadoop Disk File System (HDFS) which is based on the Google File System), is specifically designed to deal with large files [7]. MapReduce, on the other hand, simplifies many of the difficulties inherent in parallelizing data management operations across a cluster of individual machines, resulting in a simple model for distributed computing. Through the application of MapReduce, many complexities, such as data partitions, task scheduling across multiple machines, machine failure handling, and inter-machine communications, are reduced. [7]

During each run of the Hadoop job, the input gets split into multiple tasks. These tasks are effectively "atoms of work." Hadoop allows changes to the number of mapper and reducer tasks during job creation; however, once a job is created, it remains static. Tasks are assigned to "slots." Traditionally, each node is configured to have a certain number of slots for map tasks and a certain number of slots for reduce tasks, which are further configurable. Moreover, the JobTracker periodically assigns tasks to slots. Because this is done dynamically, new nodes that come online can speed up the processing of a job by providing more slots to execute tasks. Hence SLOTS_MILLIS_MAP and SLOTS_MILLIS_REDUCES represent the total time used by the map and reduce the number of tasks that are assigned to certain slots. Hadoop might schedule the same task to run on

many different nodes, so that a slow node's tasks can be completed by faster nodes if resources are scarce.

2.1.1.2 HDFS

HDFS is the Hadoop disk file system, which stores data across clusters and further provides fault tolerance by replicating data chunks across separate cluster nodes to prevent the loss of data caused by a possible single node failure, which is a very challenging task. In this way, HDFS stores multiple copies of data and provides continuous data availability.

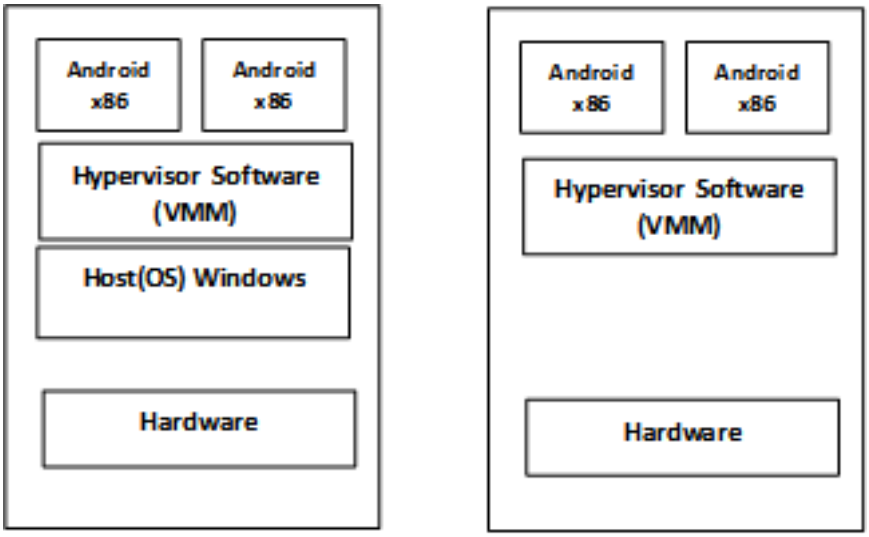
2.1.2 Cloud Virtualization

Recent progress in virtualization technology allows the use of virtual machines for real-time applications, like Voice over IP (VoIP) [34]. The term 'virtualization' broadly describes the separation of a service request from the underlying physical delivery of that service. With x86 computer virtualization, a virtualization layer is added between the hardware and the operating system. One of the observed issues is the latency and jitter resulting from buffering the network output between two virtual machine replication cycles. The virtualization approach uses either a hosted or a bare-metal architecture.

2.1.2.1 Hosted Virtualization

A hosted architecture installs and runs the virtualization layer as an application on top of an operating system, and it supports the broadest range of hardware configuration. As

shown in Figure 1(a), the hypervisor software monitors all of the virtual images configured on the top of the hosted architecture. It is responsible for allocating the resources to each of the virtual images and, hence, efficiently managing them to run optimally.



(a) Hosted Architecture **(b) Bare-Metal Architecture**

Figure 1: Hosted and Bare-Metal Cloud Virtualization Architectures

2.1.2.2 Bare-metal Virtualization

In contrast, a hypervisor (bare-metal) architecture installs the virtualization layer directly onto a clean x86-based system. A hypervisor is a virtual machine manager that allows multiple operating systems to share a single hardware host. As shown in Figure 1 (b), each operating system appears to have the host's processor, memory, and other resources to itself. However, the hypervisor is actually controlling the host processor and resources, allocating what is needed to each operating system in turn and making sure that the guest operating systems (i.e., the virtual machines) cannot disrupt one another. Since it has

direct access to the hardware resources and does not have to go through an operating system, a hypervisor architecture is more efficient than a hosted architecture, delivering greater scalability, robustness, and performance. VMware Player and Workstation are two examples hosted architectures that provide flexibility, while the ESX/ESXi server employs a hypervisor architecture that uses certified hardware for data center class performance [10].

2.1.3 Types of Virtualization

With a virtualization model integrated into an environment, one can make efficient use of such computing resources as storage, computing, and networking resources by separating the physical resource into dedicated virtual resources.

Virtualization allows multiple workloads to share a common set of resources [8]. The virtualization of the hardware infrastructure leads to an efficient utilization of computing resources by improving server utilization and consolidation, dynamic resource allocation, and management [8]. Moreover, it helps in the creation of multiple virtual images and further assists with security and automation. Finally, it also leads to on-demand resource sharing when integrated with the cloud, in the case of cloud resources.

The evolution of virtualization began with software virtualization, then progressed to hardware-enabled virtualization, then to nested page tables, and lastly to Input Output (I/O) virtualization. Software virtualization, in comparison to the other forms, is a slower and more inefficient process. It further results in poor software performance in terms of

the resource control of virtual machines, which are used to provide the virtualization infrastructure and which have proved sufficient, in terms of internal microcodes, to handle virtualization. Hence, the use of hardware virtualization extensions is needed to offload virtualization tasks from software.

To achieve virtualization, the hardware must be equipped with virtualization extensions. As of now, there are two majorly used virtualization extensions available: Intel Virtualization Technology (Intel VT) and Advance Micro Dynamics Virtualization (AMD-V). Intel VT represents a set of technologies and features that make virtualization practical by eliminating performance overheads and improving security. It also provides hardware assistance for virtualization software, reducing its size, cost, and complexity [8].

AMD-V, on the other hand, is a set of on-chip features that makes better use of and improves the performance of virtualization resources. It also provides hardware extensions for x86 architecture.[9] Although AMD's and Intel's actual implementations of processor extensions differ, both achieve the same goal of allowing a virtual machine hypervisor to run an unmodified operating system without incurring significant emulation performance penalties [10].

The virtualization of hardware resources can be broadly classified into the following categories:

1. CPU Virtualization

With CPU virtualization on x86 architecture, the Virtual Machine Manager (VMM) virtualization layer exists below the operating system layer to handle calls to the computer

hardware. The operating system is always presumed to be in total control of the computer hardware; however, in reality, this is not the case. The CPU-based virtualization could be categorized into three virtualization techniques.

Non-Virtualized Environment

In a non-virtualized environment of an x86 architecture, the CPU is in full control of the computer hardware. The overall architecture could be segmented into various protection rings, such that each ring is associated with the corresponding access level based on privileges. Ring 0 has the highest privilege, with direct access to the computer hardware. In a non-virtualized environment, Ring 0 would be the operating system, which has full control of the computer hardware. Ring 3 is a privilege layer that exists essentially for those applications that make calls to Ring 0 (where the operating system resides) [10].

In CPU virtualization, the virtualization layer is located below the operating system layer on Ring 0. The CPU instructions are trapped and made to believe they exist in Ring 0—or, in other words, that they are in full control of the computer hardware. The challenge here is that some CPU instructions or calls cannot be virtualized.

Full Virtualization

With full virtualization, the VMM runs on Ring 0, and the operating system runs between Ring 0 and Ring 3 (where the applications run). It solves the problem of non-virtualized CPU instruction by making use of binary translation. Through a combination of binary translation and direct execution, it translates the kernel code (i.e., non-modified

instructions). The resulting instructions affect the virtual hardware. The hypervisor translates these instructions to allow the application access to the virtual hardware. The guest operating system is flexible enough to run either the virtualized hardware or the physical hardware [10].

OS-Assisted Virtualization or Para-Virtualization

In this type of virtualization, the CPU instructions that are not virtualized are replaced with hypercalls (calls between the hypervisor and the operating system). Again, in this case, the operating system is unaware that it is being virtualized. In such virtualization, the guest OS is modified, as in the case of a Xen or KVM hypervisor, in which the kernel is modified [10]. The problem with this type of virtualization is that it doesn't support various operating systems.

Hardware-Assisted Virtualization

This type of virtualization makes use of the latest virtualization extensions (Intel VT and AMD-V), which improve the performance of CPU virtualization by enabling the hypervisor or the VMM layer to run in the root mode below the Ring 0 layer. This approach is different from the above-mentioned approaches, as a new layer is introduced to the x86 architecture. The new layer, sometimes referred to as the Ring -1 layer, is placed below the guest operating system (Ring 0). Placing the hypervisor below Ring 0 allows the system to automatically trap the non-virtualized CPU instructions. Hence, it does not require any OS modifications, nor does it require any binary translations [10].

2. Memory Virtualization

This type of virtualization enables the virtualization layer to allocate dedicated memory to each of the virtual machines—or in other words, on a per-VM basis. The operating system keeps mappings of virtual page numbers to physical page numbers, which are stored in page tables. All modern x86 CPUs include a memory management unit (MMU) and a translation look-aside buffer (TLB) to optimize virtual memory performance.

As shown in

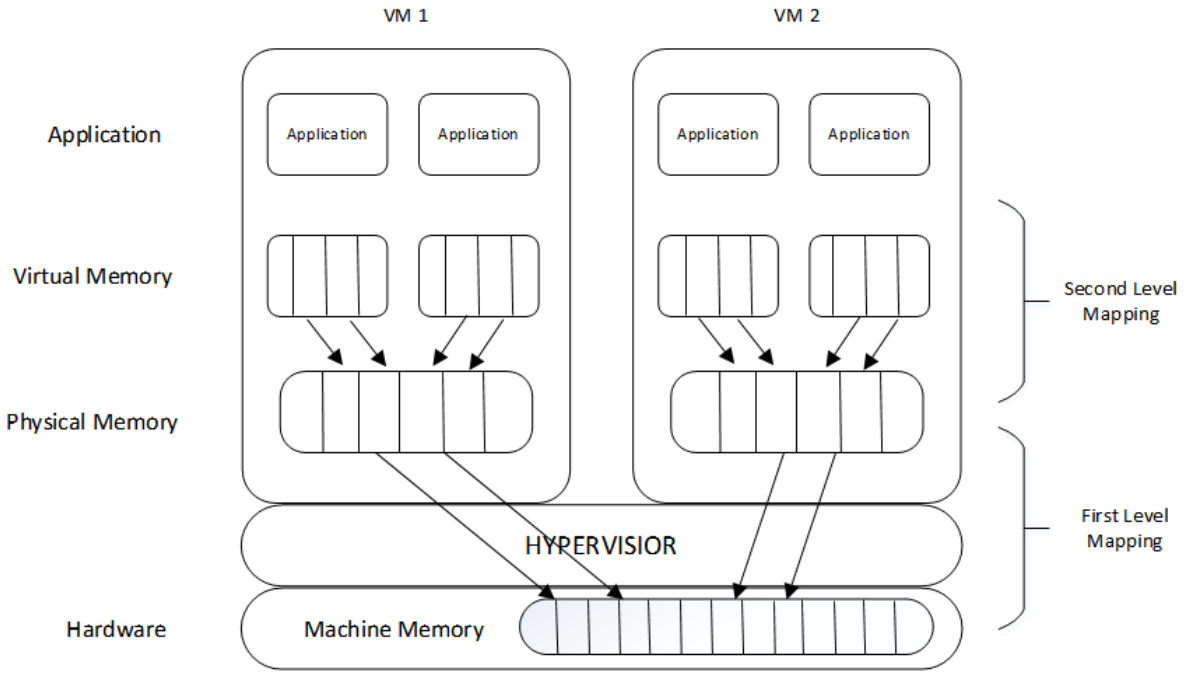


Figure **2**, the memory virtualization consists of two levels of memory mappings. In the first level of mapping, the machine memory of the hardware layer is mapped to the guest operating system's physical memory via the VMM, which uses shadow page tables to

accelerate the mappings. In the second level of memory mapping, the guest operating system's memory is further mapped to the virtual memory. The VMM is responsible for maintaining the shadow of the virtual images page table.

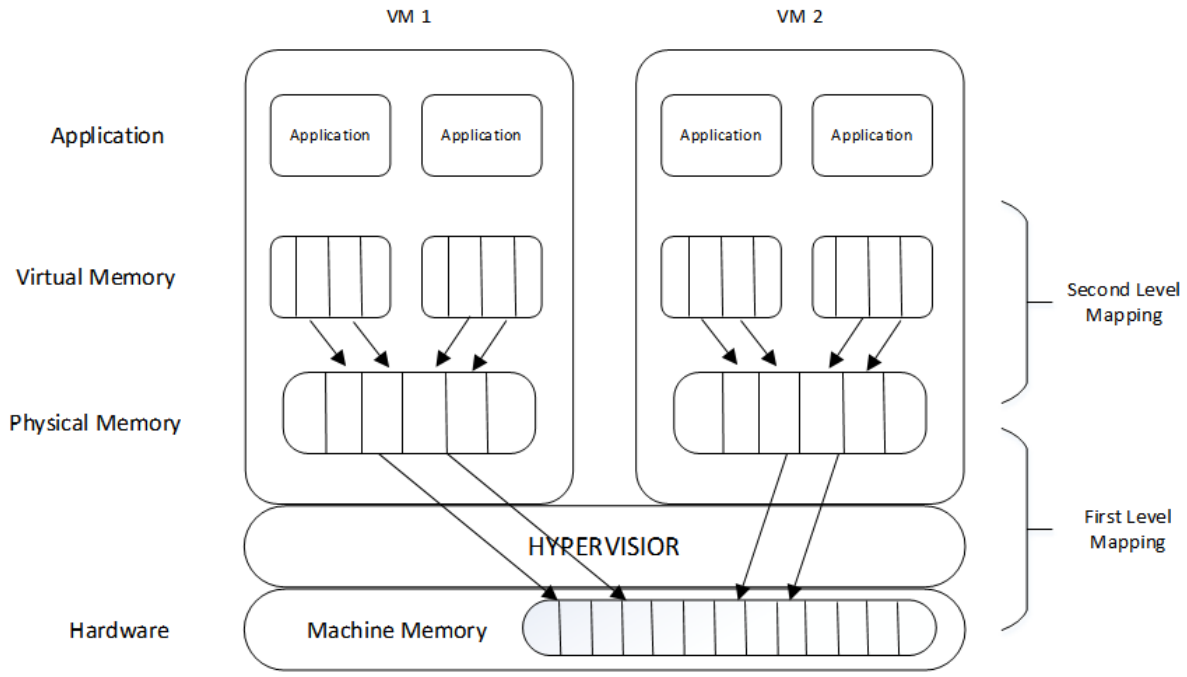


Figure 2: Memory Virtualization

The shadow page table further controls the allocation of hardware memory to each of the virtual images (VM).

3. Graphic Processing Virtualization

The graphic processing virtualization allows virtual machines to have full or partial access to the graphics processing units (GPUs), as well as to the video transcode accelerator engines. This enables such usages as remote workstation, desktop-as-a-service, media streaming, and on-line gaming usages [8].

4. I/O Hardware Virtualization

The I/O virtualization makes use of real physical device emulation to handle the I/O requests between the physical device and the guest device. The hypervisor emulates the physical device driver by creating a guest device driver in the guest operating system. The I/O virtualization could be achieved by either software-based I/O virtualization or a direct pass through virtualization. The software-based I/O virtualization makes use of the virtualization I/O stack, whose primary responsibility is to handle inter-VM communication. The virtualization I/O stack is used to multiplex the I/O request from the physical device to the virtual devices. The important aspect of I/O virtualization is to minimize added CPU utilization.

Chapter 3

Related Work

3.1 Cloud Computing Models for Healthcare Applications

The work presented [11] proposes a system called "MedCloud," which utilizes cloud-based technologies in conjunction with privacy and security rules for patients' data storage. The authors in [12] propose a mobile cloud for assistive healthcare (MoCAsH) infrastructure, which makes use of both cloud computing and collaborative plans by employing intelligent mobile agents, context-aware middleware, and collaborative protocols for efficient resource sharing and planning. It also addresses various quality-of-service issues concerning critical responses and energy consumption.

The goal of the work in [13] is to develop a cloud-assisted mobile pervasive system with medical software-as-a-service (SaaS). The mechanism of virtualization uses back-end real-time application server stacks to store and manage patients' health records. It focuses on deadline-critical real-time medical data, which are generated by sensor-based medical devices, such as wireless electrocardiograms (ECGs). In order for the system to handle time-sensitive and mission-critical medical data in a public cloud computing infrastructure, it uses a real-time application server operated as a virtual machine.

Virtualization of healthcare sensors is addressed in [14], which develops a virtual sensor for remote health monitoring applications. The system tries to overcome the discomfort issue of wearing wrist blood pressure sensors during monitoring. The application is deployed in the cloud to ensure scalability, accessibility, and flexibility. In this system, the physician or caretaker is notified immediately if there is any deviation from the normal value.

3.2 Mobile Cloud Computing Models

3.2.1 MapReduce Models and Traditional Client Server Models

Parallel processing using Hadoop's MapReduce is one solution for the faster processing of data [4]. Although parallel processing improves the efficiency of the engine, query optimization of heterogeneous data integration is one of the key issues to be resolved. However, the distribution of local data sources, as well as their autonomy and heterogeneous nature, makes these sources very difficult to optimize. We try to resolve this by integrating the SVM model with Hadoop MapReduce, which is capable of processing heterogeneous data.

3.2.2 Mobile Cloud Virtualization Models

The Cuckoo framework [2] is based on the client server communication methodology, in which the mobile device (the client) and the surrogate device (the server) communicate via the remote procedure call (RPC) and remote method invocation (RMI). The cuckoo

application is flexible enough to run either remotely or locally on an Android platform. The authors present a system to offload mobile device applications onto a cloud using a Java stub/proxy model. The Ibis High0Performance Programming System [15] is used as the basis for Cuckoo's communication component. However, the authors have not yet proposed the decision-making algorithm (i.e., regarding whether or not to offload content to the cloud); this is mentioned as part of their future work.

Some similarities and differences exist between [2] and our model. Although our model is applicable to Android and other mobile platforms and though we do support content offloading to cloud, our model does differ from that of [2] in many aspects. Firstly, we have used VNC (and its supporting protocols, RPC and RMI) instead of Ibis for performing remote operations. Although Ibis provides their model with remote access to resources and acts as a middleware, it requires prior installation and may not be suitable for frequent remote access calls. VNC, with its flexibility for running on multiple platforms and the ease with which it can port to new platforms, is an ideal fit for our scenario.

Another framework based on the client server communication architecture is Hyrax for Android smartphones, which was proposed by Marinelli [3]. Hyrax is based on Hadoop, which is ported to the Android platform. The authors present 'HyraxTube,' which is a simple, distributed mobile multimedia search and sharing program that uses MapReduce and that provides a virtualized interface to a cluster of computers to facilitate the parallel execution of tasks.

Some similarities and differences exist between [3] and our model. For example, we have used MapReduce to achieve the parallel execution of training and testing tasks (as part of the image recognition technique) in the cloud. The reason for using MapReduce in our system as to achieve faster computation for these two tasks. Although it yielded the desired results for larger training sets, the initial time taken to distribute the data and process them across resizable Amazon EC2 instances was considerably higher for smaller training sets of data. Moreover, as the previous system was based on a client-server architecture model, we had to re-write the computer vision code for it to run in a cloud-based MapReduce environment. Hence, we have opted for a cloud virtualization methodology, in which both the mobile and the cloud platform run on the same operating system (e.g., Android or another mobile platform). Since both exist on the same platform, we did not need to rewrite the computer vision code; instead, the same code was functional on both the mobile and the cloud server. Given the hardware configuration changes necessary to achieve cloud virtualization, we were able to achieve faster and better results compared to our previous model. Hence, we opted to use the virtualization methodology over the previous MapReduce task.

Unlike the above-mentioned methods, [16] adopted a type of cloud virtualization in which the mobile device decides whether to offload to the cloud or not. Once the device decides to offload, the virtual machine allocates a portion of service request processing. The authors were able to achieve reductions in average power consumption, lower service request responses, and shorter response times with virtualization. In [17], they enhance the power of mobile cloud computing by parallelizing and using multiple virtual machine

images. Using parallelization, they were able to reduce execution time and battery consumption. They also used past data to achieve energy conservation and prioritize energy usage. Although the authors have proposed several components to reduce execution time and battery consumption through parallelization, their results are based on simple algorithms, like virus scans, 8-puzzles, and face recognition, which are lighter in terms of resource consumption than the algorithms that we have used for our own application (i.e., calorie measurement and deep learning).

3.3 Conclusion of the Related Work

The virtualization methodology has been adopted by numerous frameworks, including Cloudlets, Augmented Smartphone, CloneCloud, and MobiCloud. It is clear that virtualization methodologies greatly reduce the burden on the programmer, since very little or no rewriting of applications is required [18].

Based on related work, we can conclude that, with the exception of Hyrax [3], EyeDentify [19], and Cuckoo [2], the most recent works have used either virtualization or mobile coding to offload tasks. Even the excepted projects are based on much older frameworks: Hadoop [20] and Ibis [15], which are designed for distributed and grid programming. Moreover, [18] shows that, in the area of mobile cloud computing, trends favor mobile cloud virtualization methodologies over the conventional client–server communication systems. Hence, we have adopted the virtualization methodology from our previous MapReduce model.

Chapter 4

Proposed Method

4.1 MapReduce as a Parallel Classifier for Recognizing Food Objects

In order to design a MapReduced SVM, we used Amazon's Elastic MapReduce (EMR). This method uses Hadoop to distribute data and process it across resizable Amazon EC2 instances. Hadoop uses a distributed processing architecture called MapReduce, in which a task is mapped to a set of servers for processing. The results of the computation performed by these servers is then reduced down to a single output set. One node, designated as the master node, controls the distribution of tasks to the slave nodes. The overview of the model is shown in Figure **3**.

As shown in Figure 3, a request is initially sent to Amazon EMR to start the cluster. Once Amazon EMR creates the Hadoop cluster with a master instance group and a core instance group, the master node is added to the master instance group. The slave node is added to the core instance group.

Our system deals with a huge set of images and processes them to derive results in various formats. The managing such a high volume of data can be a tedious job. Hence, to resolve such issues, we are move the processing and the

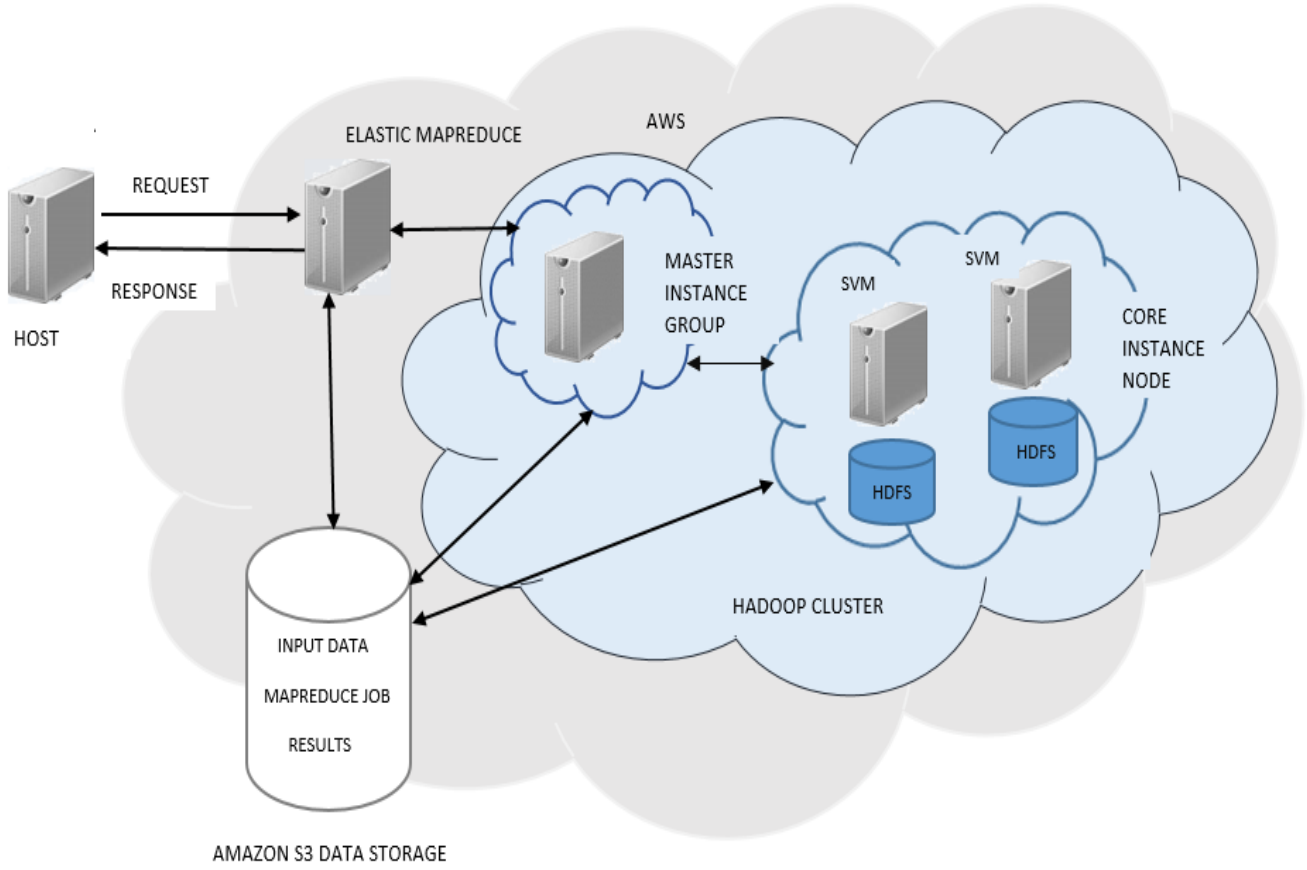


Figure 3: Hadoop Integration on Amazon EC2/AWS

storage to a cloud-based platform. The data, including the images; the non-trained, trained, and non-tested results; the logs; and the MapReduce job are all stored in the Amazon S3 bucket. Amazon S3 is storage for the Internet that is designed to make web-scale computing easier. Amazon S3 is a secure, reliable, scalable, fast, and inexpensive storage platform, making it easier for us to manage the bulk data [21].

4.2 MapReduce-based Mobile Cloud Computing Model

Since the algorithms we use involve high complexity and costs, running these algorithms on a mobile client with responsive interactions is often infeasible. Hence, we offload the processing to a high-performance server and, further, to the cloud over a network. When a user first captures a food item image using a mobile client, the image is sent to the server via Hyper Text Transfer Protocol (HTTP) (as shown in Figure 4). We use asynchronous tasks to manage our server-client communication and create a file stream for input image. We specify the URL for the image to be stored, which we achieved by installing Apache, MySQL and PHP. A PHP script on the server then invokes the server-side application to compute image segmentation on the image. Once image segmentation is run on the image, a .train file is generated by the system. After this, we train and test the data using the MapReduce SVM. [4]

The MapReduce-based cloud model is similar to Figure 3. As shown in Figure 4, a request is initially sent to the Amazon EMR to start the cluster. Once the Amazon EMR creates the Hadoop cluster with a master instance group and a core instance group, the master node is added to the master instance group. The slave node is added to the core instance group. Our system deals with a huge set of images and processes them to derive results in various formats. The managing of such a high volume of data can become a tedious job. To overcome this issue, we move the processing and storage to a cloud-based platform. Using the cloud platform not only satisfies the computational complexity

constraint of our system, but also helps us achieve a better accuracy of food recognition. The data, including all images; non-trained, trained, and non-tested results; logs; and MapReduce jobs, are stored in the Amazon S3 bucket. The food recognition test comprises four subfolders in the bucket. The job folder contains the SVM MapReduce .jar file, the data folder contains the input file (i.e., svm_data.train, svm_data.test), and the logs folder saves the log info whenever the MapReduce job is run. After the cluster is provisioned, we launch the job workflow, which involves provisioning the cluster, running the bootstrap action, running the job flow, and shutting down the cluster. Finally, once the MapReduce SVM Train job is completed, the results, including the model file, are stored in the results folder, and the logs are stored in the logs folder of the bucket. [4]

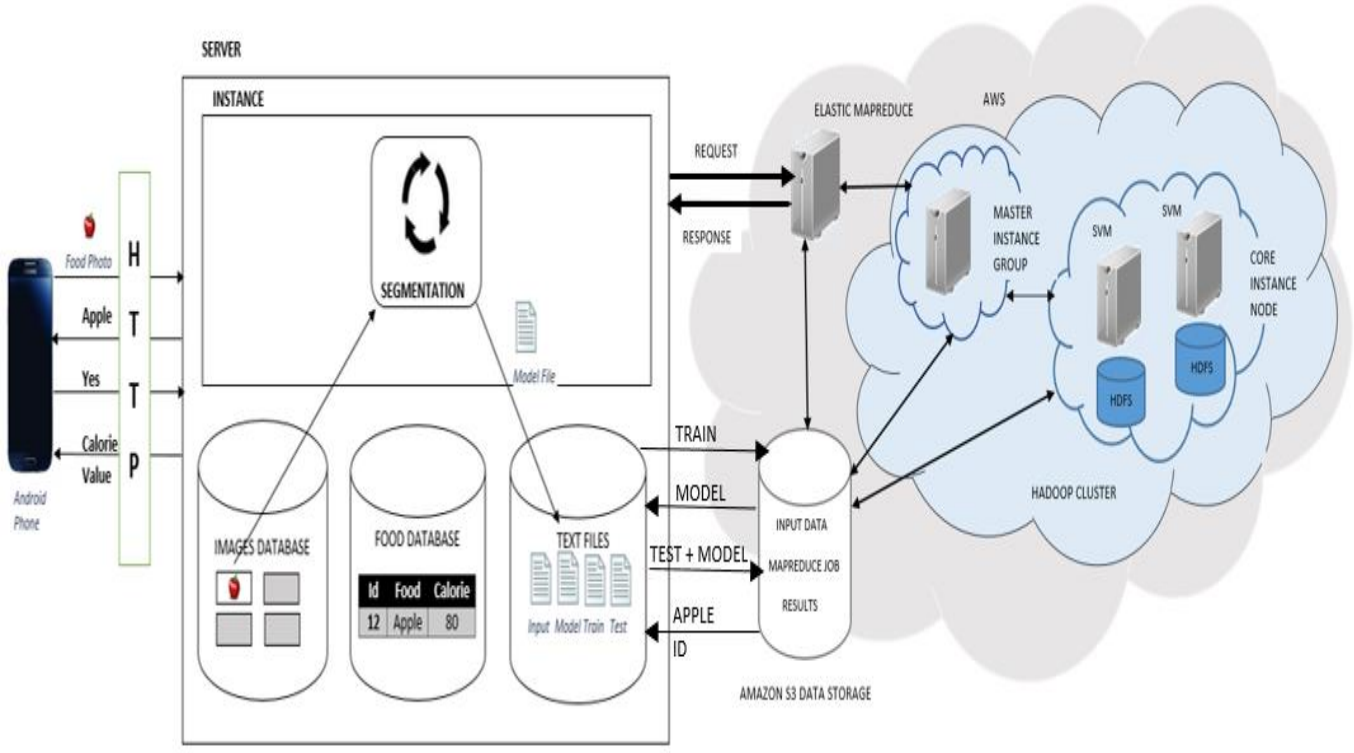


Figure 4: MapReduce-based Mobile Cloud Architecture.[4]

The trained data is then tested for accuracy by running the MRSVMTest job on the inputs. This test follows the same MapReduce procedure as the Train input job. The only difference is that the job calls svm_precision instead of svm_train. We also pass three parameters, instead of two: the input file, the output file, and the model file (as described by the arrows in Figure 4). The output contains the accuracy and the classification results. If the accuracy is lower than 90%, the test file is updated to improve future accuracy whenever we add new item to our database. If the accuracy is greater than the threshold mark, the system displays the object ID of the matching food type for that accuracy. The corresponding food type is then sent back as a prompt to the mobile device to confirm the food type details (Figure 4 illustrates prompting an apple as a food type to the user).

Finally, when the user confirms the food type, the client (i.e., the mobile device) sends the HTTP request to the server to fetch the calorie details from the MySQL database. We maintain the food database, including the food types and the corresponding calorie information, and obtain the calorie value details. We used JSON to list the corresponding calorie details for the food types on the mobile device; to accomplish this, we create a new .php file for the JSON response. While programming the Android code, an HTTP request is send to the .**php** file using a background Async task thread. After retrieving JSON from the .php file, we parse it and display the calorie value. We also create a class to retrieve JSON from a URL. This class supports two HTTP request methods—GET and POST—to retrieve JSON. Hence, we display the output with the calorie values to the user.[4]

4.2.1 Cloud Components

4.2.1.1 Support Vector Machine (SVM)

SVM classification approaches are based on hyper planes, which involve computationally concentrated processes. However, in SVM-based methods, the processing steps may become complicated, especially with growing data sets. This is the case in our system, which deals with a huge set of images (more than 3,000 images). These images need to be processed into various formats. Thus, parallel computing methods are needed to reduce system complexity and increase scalability. To apply this algorithm, we have engaged Apache Hadoop as our SVM classifier.

4.3 Cloud-Based Virtualization

4.3.1 Android and Android x86

An Android-based smartphone normally runs on an Android ARM processor, and the Android operating system is specifically built to be compatible with this processor. Android is an open source platform that includes an operating system, middleware, and key applications. It targets smartphones and other devices with limited resources [2]. Android consists mainly of Activities (which is the Java class explaining the background functionality), XML Layouts (which are responsible for the user interface of the mobile application), Content Providers, and Broadcast Receivers (for running applications at the system level), each of which has an independent lifecycle.

Android x86, on the other hand, gives us an opportunity to create a virtual Android image by using the hypervisor. This allows each virtual Android x86 image to tap into the power of the server hardware of a single host machine. The Android x86 emulator allows us prototype, develop, and test mobile applications without using a physical device. It mimics all of the hardware and software features of a typical mobile device and runs a full Android system stack down to the kernel level, which includes a set of preinstalled applications (such as the dialer) that can be accessed from the application [10].

4.3.2 KVM QEMU Cloud-Based Virtualization

Hypervisor architectures make use of hardware virtualization extensions of the server. Hence, a server needs to have virtualization extensions (Intel VT or AMD-V) enabled to achieve this. In this section, we discuss a hypervisor-based model implemented using a KVM (Kernel-based Virtual Machine), which specifically builds a bare-metal virtualization platform for a Linux host operating system. The KVM provides a full virtualization solution for Linux on Android x86 hardware containing a virtualization extension (Intel VT or AMD-V) [22].

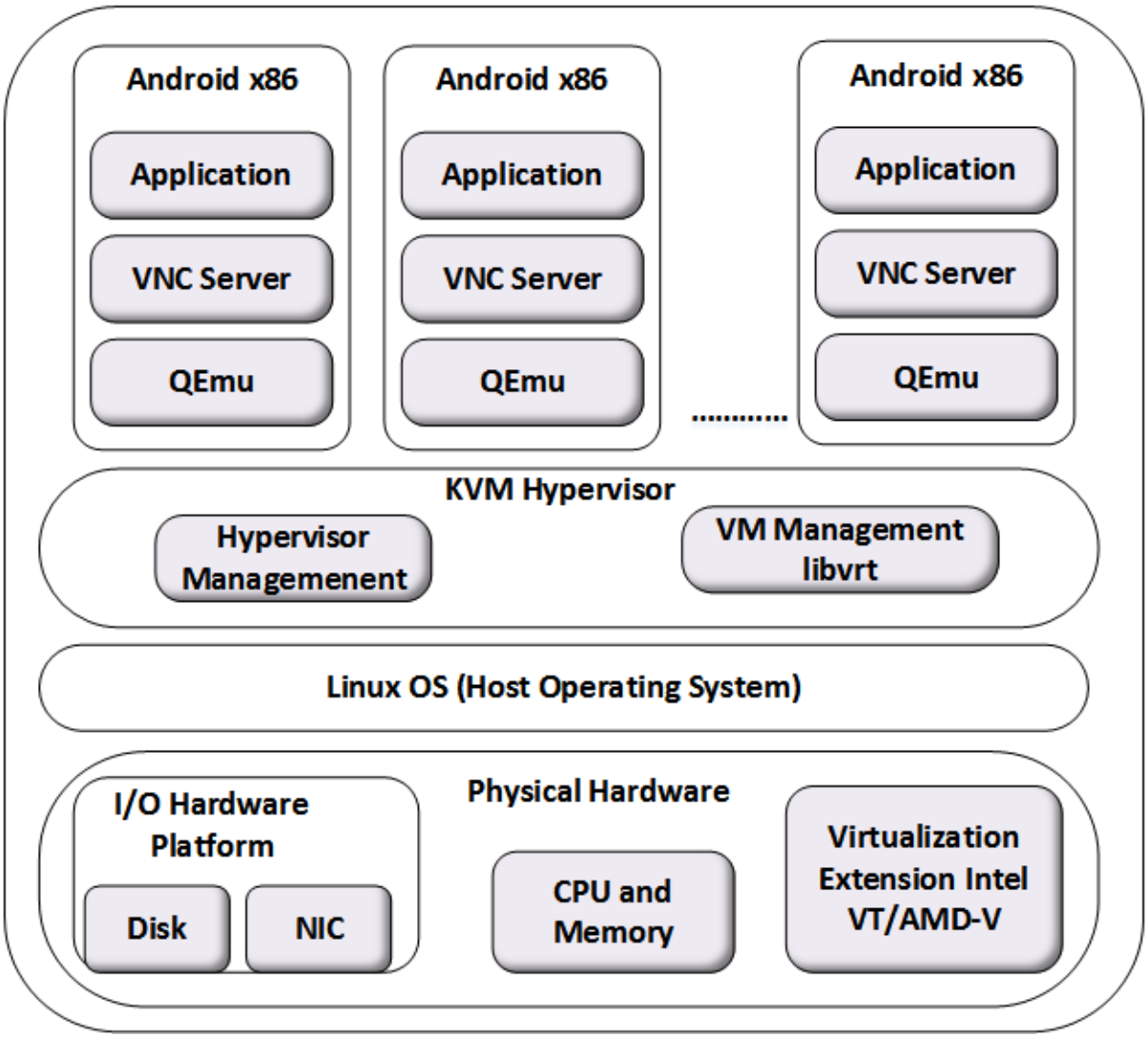


Figure 5: Cloud-based KVM Virtualization Model for EHSH.

The KVM virtualization model that we have used in our system enables us to make use of the virtualization hardware extensions of the cloud server. Our model, however, necessitates the use of Linux as the host operating system, since most of the application components (e.g., deep learning, calorie measurements, and distance measurements) have been scheduled to run as shell scripts in Linux. Hence, we adopted a model that uses Linux as its host operating system. On top of this host, we enabled virtualization extensions,

which allow us to configure Hypervisor to manage multiple Android x86 virtual images in the cloud.

Figure 5 describes the cloud-based KVM virtualization model adopted to implement our mobile e-health application. As shown, the KVM hypervisor layer makes use of the virtualization extensions (Intel VT or AMD-V) of the host Linux operating system. It then acts as the virtual image manager by managing the physical hardware resources (i.e., the CPU and memory) and distributing them across the various Android x86 virtual images.

The KVM hypervisor model primarily includes kernel modules as part of the kernel and QEMU program, which exists in every individual Android x86 image. Libvrt is used to manage the Android x86 virtual images by allocating memory and CPU resources across them.

From Figure 5, we can observe that the KVM hypervisor maps the physical hardware resources (i.e., CPU and the memory) and allocates them across each of the Android x86 images. The hypervisor also makes use of I/O virtualization (discussed in Section 4.2.1) to map the hardware I/O, including the keyboard, the mouse, the disk, and other Network Interface Card (NIC) hardware resources, across the virtual device drivers in the guest operating system (in this case, Android x86).

4.3.3 Cloud-based Virtualization Model for implementing e-Health Mobile Applications

The cloud-based virtualization model was implemented for our e-health application primarily to overcome some of the mobile phone's limitations in terms of battery capacity, network bandwidth, mobile graphics, and core processing units, as well as the e-health application's resource-intensive requirements. We were able to implement our Android-based e-health application (EHSH) in the cloud by making use of the cloud server's virtualization extensions.

By default, the Android SDK includes a virtual emulator [23] that emulates the real mobile device application for testing or implementation purposes. To implement applications on a virtual emulator, the Android SDK makes use of the AVD (Android Virtual Device) configuration. This configuration allows us to manually set the hardware configuration of the emulator by customizing the settings of the AVD. The emulator also includes some debugging capabilities, since it is capable of simulating application interrupts (such as arriving SMS messages or phone calls) and latency effects and dropouts on the data network [23].

We adopted two approaches for implementing the cloud-based virtualization model for our e-health mobile application. Both of the approaches emulate the mobile application on the emulator; however, the underlying architecture and overall performance of each one is different. The first approach is based on hardware-assisted acceleration, which makes use of the underlying virtualization extensions to run the Android emulator. This

significantly improves the performance of the emulator, which runs much faster than it would without the hardware assistance.

4.3.3.1 Virtualization Based on Hardware-Assisted Acceleration to Run Android Emulator

Hardware-assisted acceleration significantly improves the performance of the Android emulator by allowing it to run faster and making it more responsive (in comparison to a non-hardware-assisted emulator). The system could make use of graphics acceleration and CPU acceleration, which employ virtualization extensions. When the system runs the emulator using the graphics acceleration, it efficiently utilizes the Graphics Processing Unit (GPU). The graphics acceleration, however, is only supported in Android SDK Tools, Revision 17 or higher, and in Android SDK Platform API 15, Revision 3 or higher [23]. Thus, to enable graphics acceleration, we need to ensure that the GPU option is enabled.

For hardware-assisted CPU acceleration, once the Android SDK is downloaded and further installed, we can create the AVD of the Android emulator. Once the AVD configuration for the Android emulator is created and saved, the emulator can be initiated with the enable-kvm-qemu option, which enables the use of virtualization extensions to run the emulator (as shown in Figure 6). Moreover, the VNC server needs to be started and the port number needs to be mentioned when the emulator is started. However, for our model, we desired a model in which the system could create multiple virtual Android emulators, with each emulator

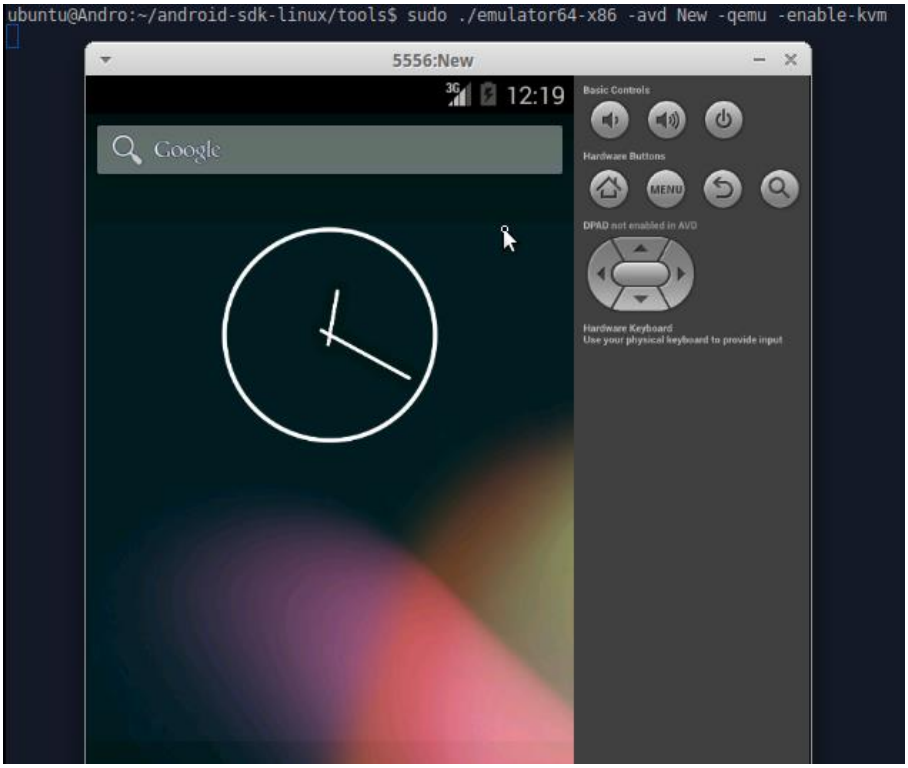


Figure 6: Hardware-Assisted Android Emulator

being assigned to a different user. This hardware-assisted acceleration is capable of efficiently running our e-health application with the virtualization extensions; however, it would not have solved the scalability concern. Hence, we opted for a virtualization-based Android x86 emulator, which facilitated the creation of multiple Android x86 images, each of which was allocated to a different user.

4.3.3.2 Virtualization Based on the Android x86 Emulator

The other way in which we implemented the Android emulator was by running the EHSH application on the Android x86 emulator. This allowed us to create multiple virtual images, each of which was allocated to a different user. As explained in Figure 5, the KVM

hypervisor provides a libvrt or virtmanager that allows us to create multiple Android x86 images (one for each user). The same could also be achieved using the command line tool, through which we could create multiple virtual images, each belonging to a single user.

As shown in Figure 7, multiple Android x86 images were created using the KVM-QEMU hypervisor on the top of Linux operating system in the cloud. In Figure 7 shows two virtual machines running, as well as their corresponding CPU usage statistics. The configuration of each of these virtual images can be adjusted by the administrator, depending on the usage of any of the Android x86 images.

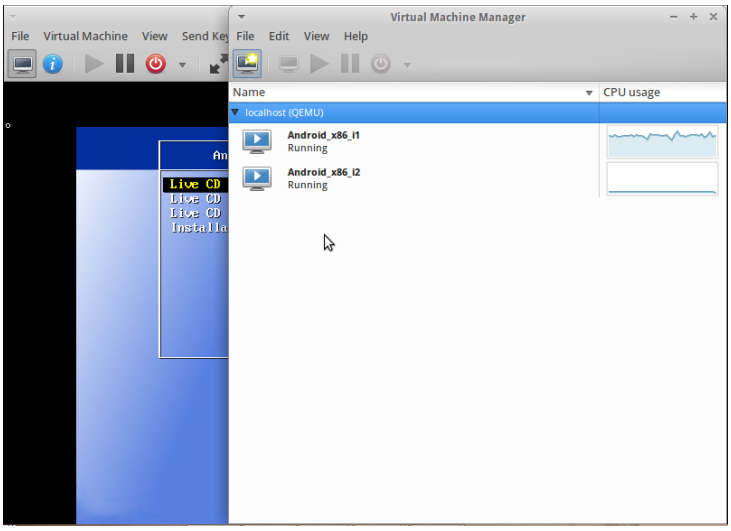


Figure 7: Android x86 Images with KVM on Linux

To change the configurations of the virtual machines, we could use either the command line tool or the VMM.

As seen in Figure 8, the virtual machine configuration parameters could be modified to promote efficient resource utilization or to improve the performance when necessary. The physical memory allocated to the virtual machine can be changed, as can the total number

of processors allocated to the virtual machine. Modifying these properties, however, would require the virtual images to be completely shut down. The configuration parameters include processor, memory, and boot options, as well as changes in the virtual network computing ports (Display VNC). Based on Figure 8, we can also observe the virtual machine performance statistics, which can be constantly monitored from the VMM.

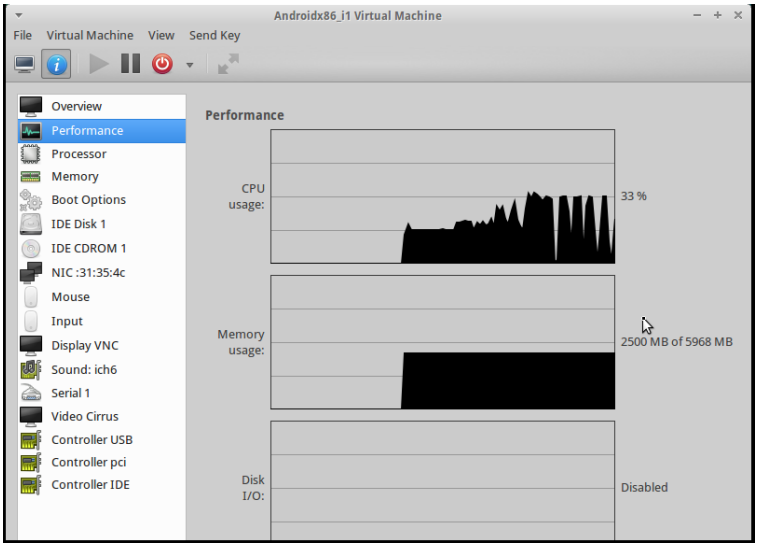


Figure 8: Android x86 Image 1 Configuration Details.

The Android x86 image (android-x86-4.4-RC1.iso) was then installed on the SDA 1 partition of the Linux operating system and was made bootable. After this step, we were able to save any consistent changes made inside the Android emulator, irrespective of the virtual machine reboot. Cloud based virtualization model that we propose for implementing our EHSH mobile application has been applied previously, but with varied implementation. Table 1, describes a list of mobile virtualization models and how our implementation differs from other virtualization models.

The architecture that we proposed is been specifically modeled for implementing our EHSH application to improve the response time, battery consumption and CPU Utilization. We implemented the KVM QEMU Virtualization in cloud environment (Amazon Web Services). The host operating system in our case, was the linux operating system, on top which we installed Android x86 emulators. We had integrated the virtualization model with the decision making algorithm and the mobile application was capable of running on hardware assisted android emulator and we compared its performance against the fully virtualized android 86 emulator. Also our goal in running the cloud based virtualization model in for our EHSH application, was to run the Food Recognition and Calorie measurement algorithm in cloud, keeping the mobile and the cloud environments on same platforms, reduce the response time in running the algorithms and achieve improved CPU Utilization and battery consumption. From Table 1 we can observe the related virtualization models where all the models have offloaded the intensive computing to the remote server (cloud or non-cloud). They address the common problem of inability of the mobile device alone to implement the computing intensive parts. These models make create the virtual images in the remote server to run certain tasks of mobile application. [35] makes use of the virtualization model to test the running of various pdf mobile application in cloud and shows battery consumption improvement. In [35] for achieving cloud based virtualization they make use of VMWare Esxi.

Table 1: Comparisons Related to Mobile Virtualization Models

Methodologies	Implementation	Virtualization Model	Virtualization Type	Operating System	Hardware assisted	Decision Making	Application	Improvements
Virtual Smartphone Over IP [35]	Mobile & Cloud	VMWare ESXi	Hypervisor Based (Full)	Android & Android x86	No	No	App. To Run large PDF documents	Battery Consumption
vNurse: Remote Health Monitoring[36]	Mobile & Non-Cloud Remote Server	Debian VM Image	Non Hypervisor (Partial)	Android & Linux	No	No	Remote Health Monitoring App.	Battery Consumption
ThinkAir [17]	Mobile & Cloud	Virtual Box & Xen Virtualization	Non Hypervisor (Partial)	Android & Android x86	No	Yes	N-queen Puzzle App.	Response Time & Energy
Paranoid Android[37]	Mobile & Cloud	KVM QEMU Virtualization	Non Hypervisor (Partial)	Android & Linux	No	No	Security Part in Cloud.	Battery Consumption
CloneCloud[38]	Mobile & Cloud	VMWare ESXi	Hypervisor Based (Full)	Android & Android x86	Yes	Yes	Image Search & Virus Scan	Energy Consumption & Execution Time
EHSH e-health Application	Mobile & Cloud	KVM QEMU Virtualization	Hypervisor Based (Full)	Android, Android x86 & Linux	Yes	Yes	Food Reco. & Calorie Measurement App.	Response Time, Battery & CPU Utilization

In [36] they run the remote health monitoring Android based mobile application for monitoring the health of elderly and illness in children that permits comprehensive, secure and modular patient remote monitoring outside a clinical environment. They make use of Debian Virtual Images for achieving virtualization in their model. In [17] they test the n-queen puzzle application and face recognition application for improved response time and

energy with the use of virtual images that were run in cloud. They make use of Xen Virtualization for achieving virtualization in their model. In [37] although they make use of KVM Qemu virtualization, they have partial virtualization for implementing the security part in the cloud, which saw and improvement in the battery consumption. [38] made use of the cloud based virtualization model to run image search and virus scan in cloud to further achieve improved energy consumption and execution time.

4.3.4 Cloud Virtualization Components

As explained in Section 2.1.2, cloud virtualization can be categorized into two main categories: bare-metal virtualization and hosted virtualization. Although we have implemented both of these methods, for implementation in the cloud, we chose hosted virtualization, due to the reasons mentioned in Section 2.1.2. The cloud virtualization components we have used in the hosted virtualization model are shown in **Table 2**.

Table 2: Virtualization Components for the e-Health Application

Components	Specifications
Host OS	The host operating system (OS) runs the Linux operating system, on top of which we installed the Android x86 images. The host operating system is the Amazon EC2 instance running in the cloud.
Guest OS	The guest OS is the Android x86 emulator, which mimics all of the hardware and software features of a typical mobile device. It runs a full Android system stack down to the kernel level, which includes a set of

	preinstalled applications (such as the dialer) that developers can access from their applications [10].
Hypervisor or VMM	For the hypervisor layer, we implemented the KVM virtualization technique. The KVM hypervisor model primarily includes the kernel modules as part of the kernel and the QEMU program, which is included in every individual Android x86 image. Libvrt is used to manage the Android x86 virtual images by allocating memory and CPU resources across them.
VNC Server & VNC Client	A VNC Server was installed on the Android x86 emulator, where it was provided rooted access. The VNC client, however, is integrated with our Android based e-health application, in which we specify the IP address and the port number of the VNC server. To ensure an additional layer of security, we also make use of a secured password, specifically to connect to the VNC server.
Android x86 Images	The Android x86 emulator was installed with the help of the hypervisor (VMM). Each of the Android x86 images was allocated to a different user. The Android x86 image exclusively ran the Android image for each of the individuals.

4.3.5 One-Way Virtual Swap

We stored a replica of our application in the Android x86 virtual machine. The running VNC client on the physical smartphone is used to connect to the VNC server on the virtual machine (as shown in the below diagram). Once this occurs, the VNC transmits all events and Android images, just like a streamed video. The VNC system allows for numerous connections, facilitating access by multiple users. To handle connections from multiple users, we assign a virtual image to each user. Hence, the user session remains consistent, and the user never realizes that the session is being run on the remote machine. The hardware virtualization (Hypervisor) exists on top of the physical hardware of the virtual machine, as shown in Figure 9, which enables the deployment of a replica VM. To maintain consistency between the Android smartphone and the Android x86 replica, any new file that is added to the file system of the smartphone (from sources other than the Internet, in the case of continuously connected operations) is also sent to the replica [24].

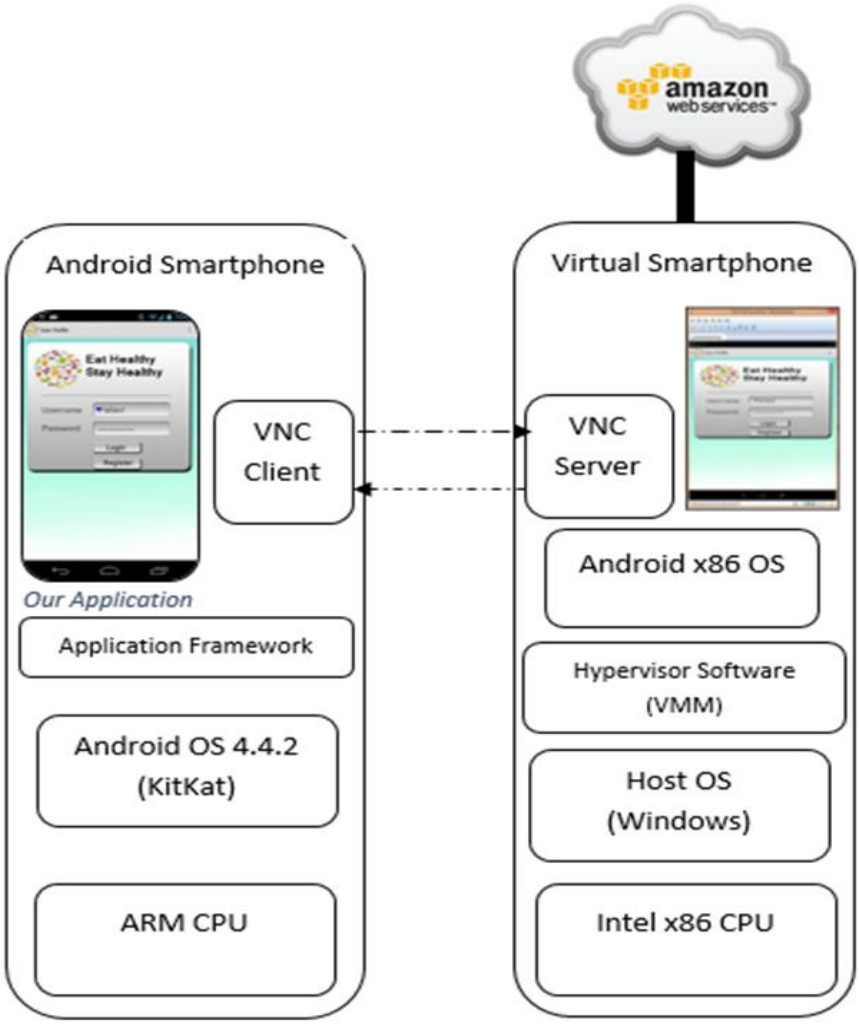


Figure 9: Implementation of One-Way Android Virtualization

4.3.6 Two-Way Virtual Swap of Foreground/Background sessions between Mobile and Cloud

In our Android application (EHSH), the user initially registers with the application and fills in his personal details, including his height (which is later used for distance calculations). As shown in Figure 10, once the user registers, he logs into the Android application with his credentials (Step 1 in Figure 10). When he clicks on the login button, multiple operations occur at the same time. Firstly, the IP address of the cloud server at the backend (not displayed on the page) and the login credentials (which are verified from the cloud

database) are sent to the cloud server. Before taking the user to the next page, the activity running on the physical device is kept in the pause mode. Meanwhile, the virtual swapping takes place, wherein the cloned session running on the cloud server is brought to the foreground, and the session currently running on the physical device is moved to the background. The user never realizes that this virtual swap takes place. To the user, it seems as though the next activity begins within the same Android application.

Once the cloud session appears on the user's physical device, it prompts him to select from the one of two stages of calorie calculation (Step 2 in Figure 10). The first step is obviously to estimate the calories of a food object, prior to user consumption. The second step is the actual calorie measurement, which will compute the calories consumed by the user. The second step (actual calorie measurement) is dependent on the first step of calorie estimation. Hence, we need to have a record of the calorie results generated from the first stage. We do this by creating a cloud virtual image for each user. Hence, each particular cloud session will only contain a replica of the related user's Android application, installed in the physical device of the user. This addresses the scalability issue of the cloud, allowing us to handle multiple users accessing the same Android application. This means that every user has an individual virtual image copy in the cloud. Every time a user logs into the application, he enters into his virtual cloud image. The swapping occurs on the user's account in the cloud. This allows us to record previous calorie results generated in the cloud, including obtaining results from the first stage (calorie estimation) and using them to measure the amount of calories consumed by the user in the second stage.

In our model (shown in Figure 10), once the user logs into the Android application, the system performs a virtual swap between the user session and the cloud session. As explained above, a dedicated virtual image in the cloud is then assigned to the user. The activity after the login page is actually running on the cloud server instance, and the user never realizes the change. When the user clicks the Estimation Calorie button, the application takes the user to the next page, where he can click on the camera button to capture a photo of the target food object (Step 3 in Figure 10). This step was challenging to implement, since, at this point, the user's physical device is displaying the content of the cloud session, but taking a photo requires the system to access the camera hardware inside the user's physical device.

Figure 10: Virtual Swapping of Foreground and Background Session between Cloud and Mobile Session.

We, thus, had two options:

1) Disconnect from the Current VNC Remote Session.

One way to access the phone's camera from the cloud server would be to disconnect the current VNC remote connection and resume the background application running on the physical device. When the user taps on the camera button, the application would access the phone's camera and allow the user to capture the image. It would later re-establish the remote VNC connection back to the cloud session. However, this option would impede our efforts in two ways: First, it would increase the overhead of the Android application by establishing a remote VNC connection in the first stage, then disconnecting the connection, and later reconnecting again. This defeats the purpose of using cloud virtualization, whose primary functionality is to reduce the overhead on the user's physical device. Second, this approach precludes us from introducing new features to the camera activity, which is an integral part of our distance calculation step. Based on these factors, we propose an alternative approach, explained below.

2) Switch between Foreground and Background Session.

Since two sessions of the Android application (one session in the cloud and the other in the mobile phone) are present, we can switch between these sessions without disconnecting from the VNC. When the user logs into the Android application, he is connected to the cloud session. In other words, the cloud session becomes the foreground session. The session that was previously running on the user's mobile device, meanwhile, is switched to the background. When the user clicks on the camera button in the cloud session, the system

switches the background session (running on the user's mobile phone) to the foreground. In this way, we are able to introduce our camera activity from the cloud by accessing the live-feed from the camera hardware of the user's physical device. If we explore this scenario more closely, it is clear that this method enables the Android application to switch back and forth between the cloud and the mobile session, without affecting the user's mobile experience.

4.3.7 Android-Based e-Health Application (EHSH) Components

As shown in Figure 11, the Android-based e-health application contains two major cloud components: a virtual computer architecture (explained in Section 4.2.6) and a deep learning algorithm. As shown in Figure 11, the Android-based e-health application takes food images as input, processes them (in the cloud), and displays the calorie result.

Figure 11: Cloud Components of e-Health Application (EHSH) [24]

4.3.7.1 Deep Learning

Deep structured learning—or, more commonly, "deep learning" or "hierarchical learning"—has emerged as a new area of machine learning research [25]. Deep learning involves learning multiple levels of representation and abstraction, which help to make sense of data, such as images, sound, and text. [25]

For our multimedia e-health application, we use deep learning as the machine learning algorithm to extract food features based on the color, contour, texture, and size of the food object image. We then further classify the food object based on the food object name. Stochastic gradient descent (SGD) [26] algorithms were used to train the deep learning network. Using the decision algorithm, we have implemented the deep learning algorithm in multiple cloud instances. Since the algorithm is highly resource intensive, we offload the computation to the cloud. There are two stages involved in classifying a food object with the deep learning method. Specifically, once we trained our food images with the deep neural network, we generated the model file, performed the segmentation, and extracted the features, which were further written into hidden layers in the deep network (the libraries and model files of the deep neural network here are based on the work of Hinton [27] and Srivastava [28]). After customizing the top-level feature layer, we were able to generate classification results [29].

4.3.7.2 Calorie Measurement

To compute calories, the mobile e-health application makes use of two calorie measurement techniques, which enable the application to accurately determine calorie content. For one of the approaches, we use smartphone sensors to determine the distance between the phone and the food object. This gives the system a reference point from which to calculate the calorie content [29]. The distance is further applied to the block resize algorithm [29], which formulates blocks on the food image and further recalibrates the block sizes based on the distance from which the food image is captured. This approach

enables the system to intelligently calculate the calorie value for each food object, without the need for any reference object (i.e., an object with known dimensions), thereby differentiating this method from other approaches. The block size algorithm is then used to calculate the perimeter and area of the food portion in the image, which is finally used to measure the calorie content. We use this approach to compute the calories of food objects, irrespective of how far the food objects are placed from the phone.

The other approach for computing calories is based on the concept of a reference object (in this case, the thumb of the user). The dimensions of the user's thumb are initially recorded, which enables the system to calibrate the dimensions of food portions based on photographs taken of the food object from the top and the sides. The dimensions further enable the system to calculate the volume of the three-dimensional food object, which is then used to compute the corresponding weight of the food object. Based on this information, we are able to measure the calorie content of the food portion on the plate [6].

Chapter 5

Implementation Design

We will discuss the two approaches that we implement as part of the distributed cloud computing approach on the mobile device. Both of these models are completely different in terms of how they are implemented, although each one's prime functionality is to provide distributed cloud computing to implement a resource-intensive task and remove overhead from a mobile device.

5.1 Implementation of MapReduce Train and MapReduce Test on Amazon EC2 Instances

In this section, we will discuss in detail how MapReduce was implemented to run the support vector machine algorithm, in parallel, in various cloud instances. We were able to implement the MapReduce Train and Test models in the cloud using the proposed method, as explained in the previous section.

5.1.1 Software & Development Configurations

Hadoop 0.19.1, LibSVM-3.17, Eclipse Europa, and Cygwin Terminal were used to implement the MapReduce SVM program on the single node cluster.

Hadoop 1.0.3, LibSVM-3.17, Eclipse Europa, Amazon EMR, Python, Ruby 1.8.7, RubyGems version 1.8, Amazon EC2, Amazon S3, and Amazon EMR were used to implement the MapReduce SVM program on the multiple node cluster in the cloud environment (in our project, we implemented Amazon Web Services).

5.1.2 MapReduce Implementation

The implementation of MapReduce for the SVM model can be broken into the following steps: compute statistics for features (color, size, shape, etc.) and class objects, transform the sample by implementing the SVM model, compute statistics for a new feature space, and finally distribute the new samples and train the model in a random order with the reducer function. The SVM model is implemented in parallel with the help of the MapReduce mechanism, in which each instance is trained with an SVM model. The support vector of each subSVM is taken as an input for the next layer subSVM [30]. The non-support vectors are filtered with the subSVMs.

From a Hadoop MapReduce perspective, the data-splitting strategy can be accomplished according to the number of MapReduce tasks that will be employed. Each Map task (MAP1…MAPn) will process an associated data chunk (DataChunk1…DataChunkn) and generate a respective set of support vectors (SVset1…SVsetn). These can then be forwarded to a single (or multiple) reducer (REDUCE1), which will contribute the feature elements of the respectively aggregated support vector (SV) set of the global SVM to a final learned model [31].

Initially, we planned to test our code in Amazon EMR; however, due to the extensive cost and timing issues, we developed an alternative solution. We attempted to avoid this problem by copying the data to HDFS, processing it locally, and uploading the output back to Amazon's S3. This saved us time and money for training and testing the data.

Table 3: Steps to Implement the MapReduceSVM code on the local host.

Implement the MapReduceSVM code on the local host. **Configuring the SSH** - Install Cygwin and Eclipse on the local host. - Configure the SSH daemon; start the SSH daemon. - Generate the authorization keys and check the connection to the local host. **Setting up the Hadoop Disk File System(Hadoop DFS)** Installing and Configuring Hadoop 0.19.1 - Download Hadoop 0.19.1 and untar the files to the home directory folder. - Install the Java jdk and configure the environment variable for Hadoop. - Configure the directory where Hadoop will store its data files. Create three configuration files: core-site.xml, mapred-site.xml, and hdfs-site.xml.
Starting the Cluster - Format the HDFS filesystem via the Namenode.

- Start the single node cluster by starting the Namenode, Datanode, JobTracker, and TaskTracker on the system.
- Check the status of the Hadoop process using the jps tool (part of Sun's Java v1.5.0).
- Check with Netstat to determine whether Hadoop is listening to configured ports.

Uploading Files to HDFS

- Install the Hadoop Plugin in Cygwin by copying hadoop-0.19.1-eclipse-plugin.jar from Hadoop to the Eclipse folder.
- Open Eclipse; the Map/Reduce file should be visible in the list of perspectives.
- Place the MapReduce SVM job and the input data on the HDFS.
- Verify whether the files were uploaded correctly by opening the Eclipse environment. The files should be displayed under the DFS locations folder in Project Explorer tab of the MapReduce perspective.

5.2 Implementation of the Virtualization Framework in a Mobile Environment

5.2.1 Hardware Configurations

Although the system does not require the physical and the virtual smartphones to be on the same platform, this particular setting allowed us to more tightly integrate both environments. Hence, we configured our system in such a way that both the smartphone and the virtual machine's operating systems were Android. The smartphone that we used

was the Samsung Galaxy S4, which has been upgraded to Android 4.4.2 (KitKat). The Android x86 emulator used an Android 4.4 (Kitkat) image. The configuration of our physical device Samsung Galaxy S4 included an octa-core processor and a 1.6 GHz Quad + 1,2 GHz Quad CPU speed with 2GB of RAM.

Before running Android x86, we needed to ensure that the system's CPU would support one of the following virtualization extension technologies: Intel Virtualization Technology (VT, VT-x, vmx) or AMD Virtualization (AMD-V, SVM) (only supported for Linux). If not, then we would need to enable virtualization extensions in BIOS and confirm that the host operating system sensed it. To deploy our application, we used the Android SDK. The Android SDK Manager was used to install the following components: Android SDK Tools, Revision 17 or higher, and an Android x86-based system image. [32]

5.2.2 Flow Charts of the Cloud Based Virtualization Model

To enable the hardware-assisted cloud virtualization, we refer to **Figure 12**, which depicts the hardware-assisted cloud-based KVM virtualization. As shown, the first step would be to check whether the virtualization CPU extension is enabled, in which case we would proceed to verify the KVM modules. If the CPU extension is not enabled, then we enable it from the system BIOS settings. To enable the KVM module, the source of the hardware virtualization (i.e., Intel or AMD) is important. Then, we load the appropriate KVM modules to achieve hardware virtualization.

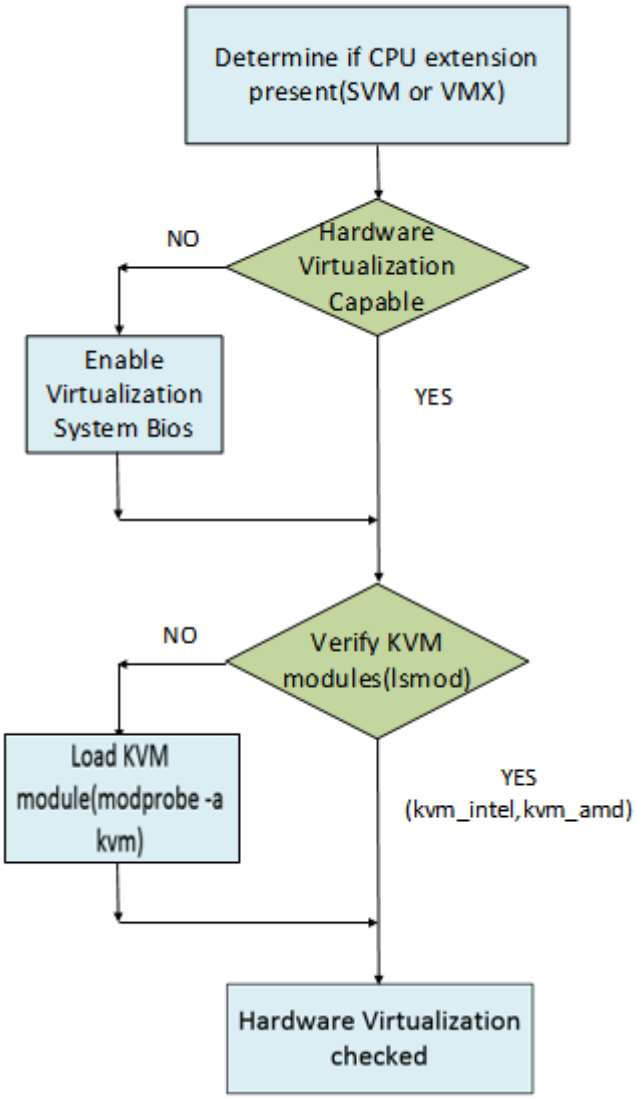


Figure 12: Flow Chart for Enabling Hardware-Assisted KVM Virtualization.

From F**igure 13**, we can observe that, once a hardware CPU extension is verified, the system does a graphic acceleration check and verifies whether the graphic acceleration is enabled. If it is enabled, the system then proceeds to install the VNC server on the cloud

server. Once the vnc4sever is installed, we set the display and port settings and finally direct the VNC client to run the port on which the VNC server is running.

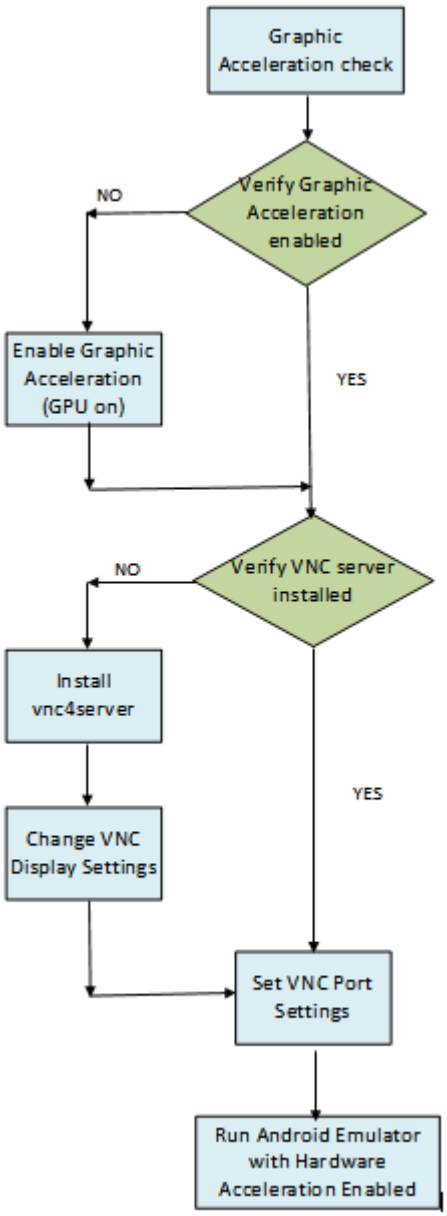


Figure 13: Flow Chart for Running Android Emulator with Hardware Acceleration

5.2.3 Decision Making Algorithm and Multiple Object Detection

The cloud server is responsible for running multiple virtual images, each containing replicas of mobile sessions of different users. It also acts as an intermediately server—or, in other words, a cloud broker—that runs the decision-making algorithm. The intelligent cloud server then performs a preliminary examination of the photo captured by the user. It analyzes the image to separate the distance information and performs segmentation to split the images and obtain a count of the food objects on the plate. The total count is fetched, irrespective of where the food object is placed on the plate. Since deep learning is a heavy computing algorithm, we implement it on a different cloud server. Specifically, we use three sets of cloud servers—the t2.small (S), t.medium (M), and x3.large (L) instances from Amazon Web Services (as shown in Figure 14). Although each of these servers is capable of running the deep learning algorithm, we integrate them to achieve scalability, cost optimization, and parallel processing. We built the architecture based on the consideration that the total time needed for image processing and calorie computation remains consistent, irrespective of the number of food objects on a plate or the number of users connecting to the application.

Once it has a count, the decision-making algorithm sends images to the other cloud servers to perform image processing using deep learning. The algorithm follows two parameters for sending images to cloud servers: 1) the total number of food objects and 2) the performance statistics of the cloud servers. If there are two or more objects, the system will prefer the (M) and (L) cloud servers for image processing. For images containing only

one food object, the system will prefer the (S) cloud instance. To obtain the performance stats, the system consistently monitors the statistics of the cloud server in terms of CPU utilization, space availability, cache size availability, and whether the cloud server will be able to quickly and efficiently handle the computation power. Since the cloud servers operate on a Linux operating system, we performed preliminary Linux-based scripts for health checks on the server. these included the top command for checking CPU utilization and the memory consumption. We set different thresholds for the CPU utilization; for example, if CPU utilization were to exceed 85%, then the decision algorithm would not send any images to the server. If the CPU utilization were to exceed 60%, then the server would be used for processing. Similar thresholds were set for disk space usage, such that if the disk usage exceeded 90%, then the decision algorithm would not send any further images to the server for processing. Based on these factors, the cloud broker is able to determine whether any of the cloud servers are currently busy. It then evaluates the logs, which contain the performance statistics of the cloud servers. Based on this evaluation, it will conclude whether the target cloud servers are busy or free to perform the image processing.

To enable computation in the cloud, we have strategically divided the computation algorithms (as explained above) across different cloud server instances. We did this while keeping in mind, firstly, the total response time needed to generate the final calorie count; secondly, the number of users connecting to the cloud; and, thirdly, the need to optimize the cloud usage in such a manner that it efficiently uses cloud resources, while minimizing the cost of cloud instances. Costs for cloud instances, especially through Amazon Web

Services, follow a pay-per-use model. Hence, we will only be charged for those instances that are active and being used (rather than being charged a monthly fee for the cloud servers). This gives us the flexibility to add or remove any number of instances in real time, when necessary according to the computation. For example, in scenarios involving increased numbers of user connections, the instances would accordingly increase. At the same time, this approach also allows us to shut down the cloud servers when not in use.

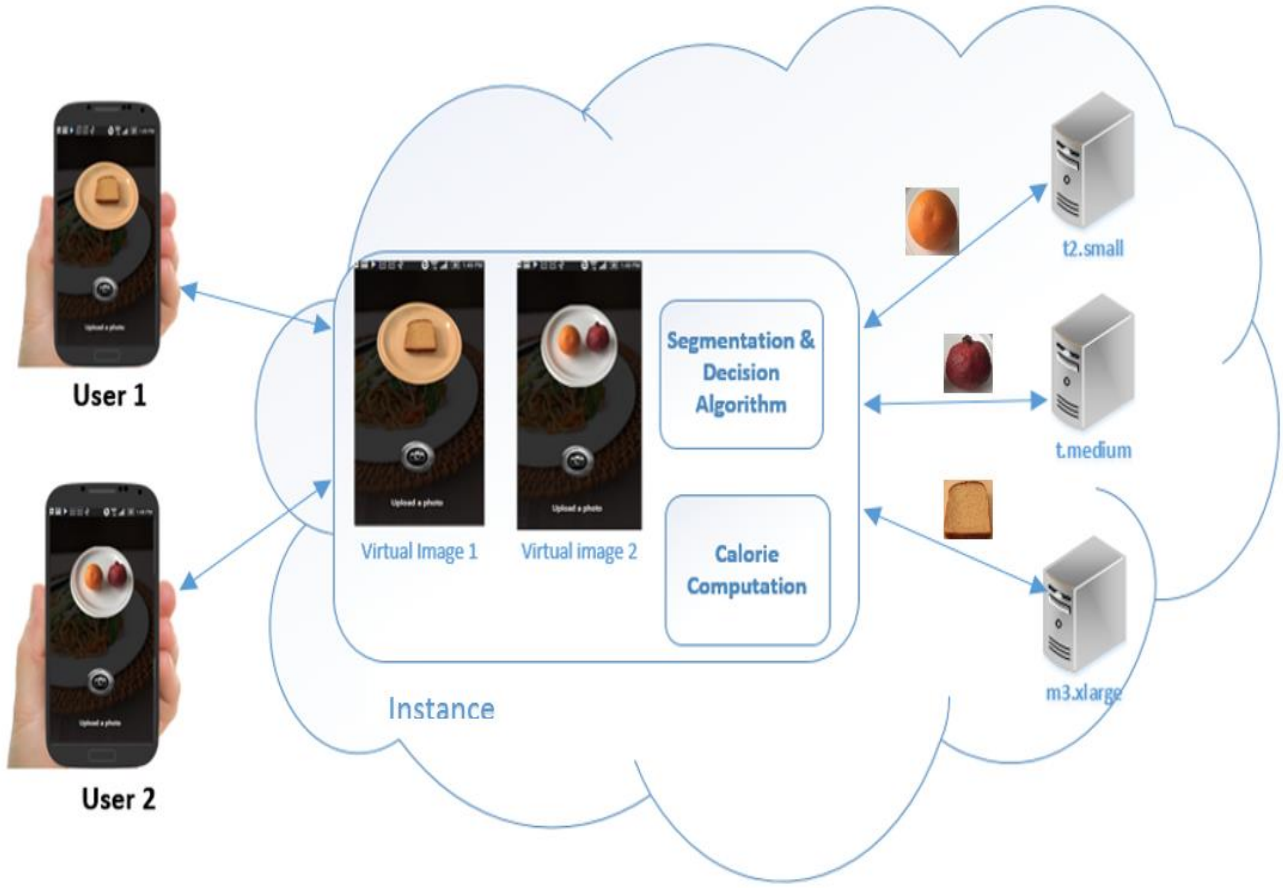


Figure 14: Mobile Cloud Virtualization Architecture

As shown in Figure 14, when two users attempt to connect to the cloud server at the same time, the cloud server creates a virtual image for each user. Once a user captures an image

of the target food object and the cloud session remotely accesses this image, along with the distance (i.e., between the user and the food object), the system proceeds to the image processing stage. The image processing is based on the deep learning model, which, when computed along with a calorie computation algorithm, necessitated increased processing. We wanted to avoid running both of these algorithms on the same cloud instance, since this cloud instance would already be running the Android virtual image of the mobile device. Hence, we introduced another layer of cloud instances, whose primary functionality was to perform image processing and calorie computation algorithms. In this layer, we used three cloud servers, each with a different processing capability (t2.small (S), t.medium (M) and x3.large (L)). In this scenario (as shown in Figure 14), User 1 captures a photo of bread, and user 2 captures a photo of a pomegranate and an orange. The cloned server, which is running the virtual replica image of the Android application (EHSH), will then decide between the above-mentioned cloud servers, based on factors that will help it make a smart decision. The system segments the food objects and sends them to t2.small, t.medium, and m3.large. This solves the problems of load balancing, resource optimization, scalability, and debugging. The decision is based on the number of food objects in a plate, a performance statistics check of the cloud servers and an analysis of previous scenarios from the database. To obtain the total number of food objects in the plate, we perform image segmentation, which helps us determine the total count of food objects on the plate. Through this process, the food object is split into multiple images, which are sent to different cloud server based on our decision algorithm. We have prioritized the sequence in

which the system performs the analysis to make the decision. The first step is to count the total number of food objects on the plate.

Chapter 6

Evaluation Result

6.1 MapReduce Results

We implemented the MapReduce method to achieve distributed cloud computing by introducing parallel classifiers across each of the cloud instances. Furthermore, we calculated the overall time used by the MapReduce task across each of the cloud nodes and further studied the time used by individual components of the proposed method (Section 4.1).

6.2 Processing Time of Various Cloud Instances

Using the latest version of Hadoop, we were able to obtain the amounts of time that mappers and reducers spent executing in aggregate (SLOTS_MILLIS_MAPS and SLOTS_MILLIS_REDUCES). However, this time value included time spent on failed tasks. (the larger the cluster, the more likely that individual jobs will fail, requiring a restart of that part of the processing) [33]. The map tasks and the reduce tasks significantly contributed to the total time. Hence, we follow the below notations, which describe the time taken by each of these two tasks:

1. SLOTS MILLIS MAPS: Total execution time of all the maps taken together.

2. SLOTS_MILLIS_REDUCES: Total execution time of all the reducers taken together.

Firstly, the input data is analyzed with only three computation node clusters (i.e., a parallel SVM method with MapReduce analyzed with one master node and two core nodes is used to train the SVM model). The trained model is used to predict the testing samples. SLOTS_MILLIS_MAPS and SLOTS_MILLIS_REDUCES are listed in Figure 15. For comparison, the sample is partitioned into three, four, five, and six sub-samples. When the sample is partitioned into three sub-samples, three computing nodes are used.[21]

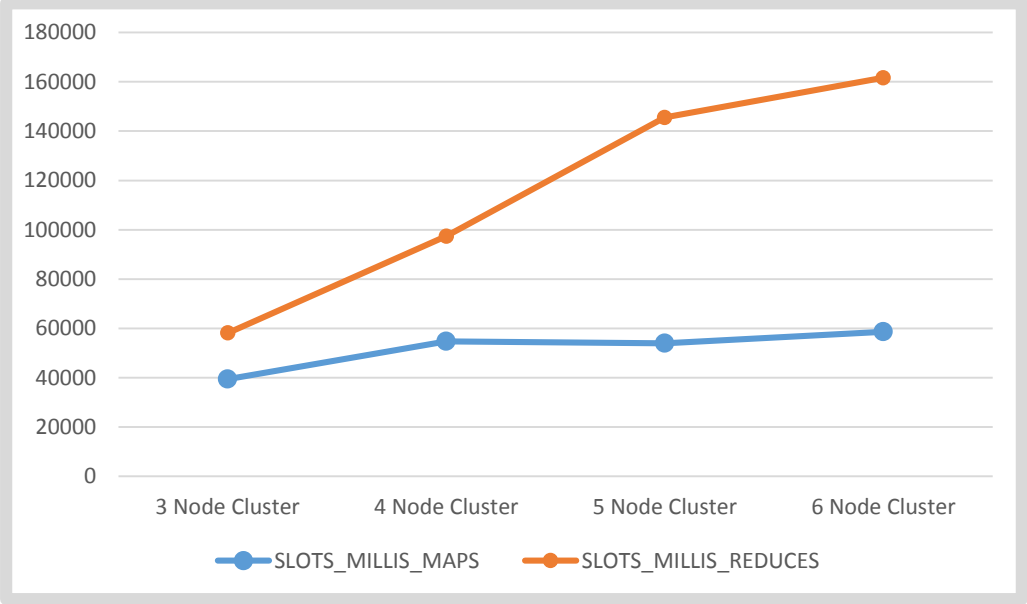


Figure 15: SLOTS_MILLIS_MAPS and SLOTS_MILLIS_REDUCES based on different cluster sets (Time in Milliseconds v/s No. of Node Cluster) [21].

As shown in Figure 15, the time taken by the map tasks and reduce tasks, increased with the increasing number of nodes in the cluster, although the map time was reduced when

switching from 4 node cluster to 5 node cluster. Hence the time taken by the map and the reduce tasks doesn't proportionally increase with the increasing nodes in the cluster. To reduce the total time, it is imperative to use the right combination of the cluster nodes that would reduce overall response time, in this case it being cluster of 3 nodes. The runtime reported by the system is a measure of the execution time from start to finish, which is influenced by a number of factors, including (in particular) the cluster load. Depending on the availability of a cluster the individual map and reduce tasks may require different periods of time in the pending state (i.e., the state during which they are waiting to be executed). Therefore, we report the actual run time of the mapreduce jar file, without considering the provisioning and bootstrapping steps [21].

The time spent by reducers can be decreased simply by increasing the number of reducers, which is a user-configurable parameter. The number of reducers also determines the number of sub-indices that will be generated [33].

Table 4: Comparison of Time Used by MapReduce for Various Cluster Nodes [21].

Tasks	3 Node Cluster	4 Node Cluster	5 Node Cluster	6 Node Cluster
Default Number of Map Tasks	8	12	16	20
Default Number of Reduce Tasks	3	5	7	8
CPU Time Spent	7.440s	8.100s	10.200s	11.010s

(s)				
Overall Time Taken for MapReduce Task.	180s	171s	155s	148s

6.3 Processing Time based on Cloud Virtualization & Intelligent Decision Mechanism

The setup of our testing was as follows: We used seven different food classes, each containing 20 test images. For each image belonging to a class, we record the recognition accuracy, the recognition success, and the time (in seconds) to process the results to the user. We run the first set of experiments on a single-instance cloud server (1 ECU and 2GB RAM), in which we include the top five image results. The average timings for the implementation of the above algorithm were recorded between 16 and 18 seconds. For most of the image classes, the recognition success rate was 18 out of 20 images, with accuracy varying between 59.05% and 99.99%. Except for a couple of images, the recognition rate of most food items was above 95% [21].

In the second setup, we performed the experiments on a cloud server with two instances, each with 4 GB RAM and 3 ECUs. For this experiment, we included the top five image results. The average timings for the implementation of the above algorithm were recorded between 15 and 16 seconds. The recognition success rates were similar to those in the case of one instance cloud server, and the accuracy rates were between 59.05% and

99.9487%. In the third setup, we performed the experiment using four instances of the cloud server with 15GB RAM and 13 ECU. With this relatively powerful cloud server, there was significant improvement in the timings, and the average processing time was 14.60 seconds[21].

Table 5: Comparison of the results of the application processing times [21]

		Time (second)			
		One instance cloud server (2GB and EBS storage)	**Two instances cloud server (4GB and EBS storage)**	**Four instances cloud server (15GB and EBS storage)**	**Local Server (1.7 GB RAM)**
Food Name	**Spaghetti**	17.61	15.85	14.62	24.2
	Burger	17.47	16.07	14.6	27.41
	Bread Slice	17.55	15.28	14.675	27.09
	Banana	17.92	16.36	14.68	26.71
	Strawberry	17.6	15.7	14.6	27.39
	Pineapple	17.46	15.14	14.68	27.2
	Cucumber	16.60	15.03	14.63	28.77

The experiment was also performed on the local server, with 1.7GHz and a shared 1.78 GB RAM. We noticed a significant difference with respect to the time and the recognition results in this case. The average timing for the implementation of the algorithm ranged between 24 to 28 seconds, with varying results on each run. Moreover, only 16 of 20 images were recognized.

The results of the processing times for the above experiments are shown in **Table 5**. As we can see, the time improved significantly for all cloud configurations compared to the

local server. As expected, the time improvement increased as the processing power of the cloud sever increased. For some food items, the boost in time processing reached 50% percent improvement, as shown in **Table 6Table 6**. One aspect of the results that did not exhibit significant improvement was the accuracy of the food recognition. Even when running the system on the most powerful cloud server, there was no improvement here. The reason for this result is the number of images used in the experiment. The training algorithm requires a much larger data set to illustrate the difference between the two experiments. This is a limitation of the current experiment; however, we plan to add more images in further investigations [21].

Table 6: Percentage improvement in processing times [21]

		Percentage improvement in processing times		
		One instance cloud server s. local server	**Two instances cloud server vs. local server**	**Four instances cloud server vs. local server**
Food Name	**Spaghetti**	27%	35%	40%
	Burger	36%	41%	47%
	Bread Slice	35%	44%	46%
	Banana	33%	39%	45%
	Strawberry	36%	43%	47%
	Pineapple	36%	44%	46%
	Cucumber	42%	48%	49%

As shown in Figure 16, we observe improved timings for the five food classes. For example, the total time for a calorie measurement of a pomegranate and an orange is 17 seconds, which is improved to 13.7 seconds (20.5% improvement in processing time) through the use of a decision algorithm. Similar time improvements were observed for all multiple-food instances, such as the case of two bananas (for which an improvement of 16.6% was observed) and the case of two slices of bread (for which a 29.4% improvement was observed), with the use of the intelligent decision algorithm. Based on these results, we can deduce that the decision algorithm is able to reduce the overall time consumption required for calorie measurements in the scenario of multiple-food objects. In the case of a pomegranate and an orange, the results were based on the scenario explained in Figure 14, wherein two users were simultaneously accessing the system. The Intelligent Decision Mechanism was able to segment the food object image into two images. These were then sent to m3.large and t.medium by determining that there were two food objects and that the performance statistics of the both servers were below threshold levels. For single food objects on a plate, we used the t2.small (S) instance, as mentioned in the decision algorithm description above. We were, hence, able to achieve similar timings for single and multiple food objects, as well as to optimize cost (since we used the cheapest cloud instance based on the decision algorithm).

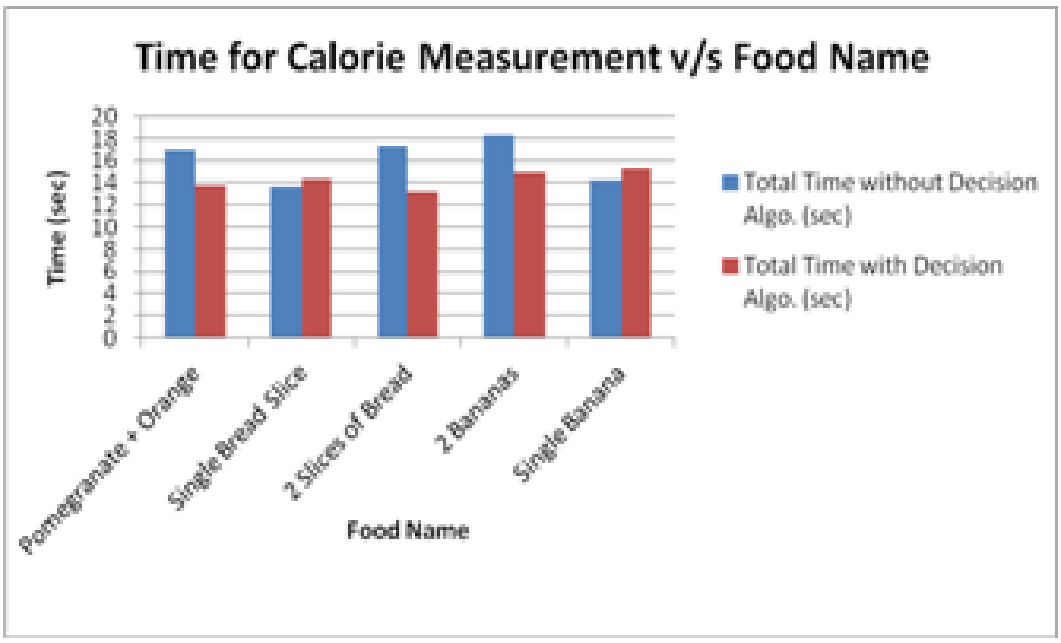


Figure 16: Time Comparison between Calorie Measurement vs. Food Name.

By using the cloud virtualization model, we observed that the decision algorithm takes the least time when compared to the other algorithms as the average time taken for the decision algorithm was 1 sec. The deep learning algorithm (as shown in Figure 17) has relatively higher time consumption with 6.1 sec which improved significantly because of the decision making algorithm, when compared to other methods. Because of using the decision algorithm, the timing results for deep learning and in turn the final calorie computation improved significantly by 20.5%. Hence each of these methods have significantly contributed in improving the overall time consumption.

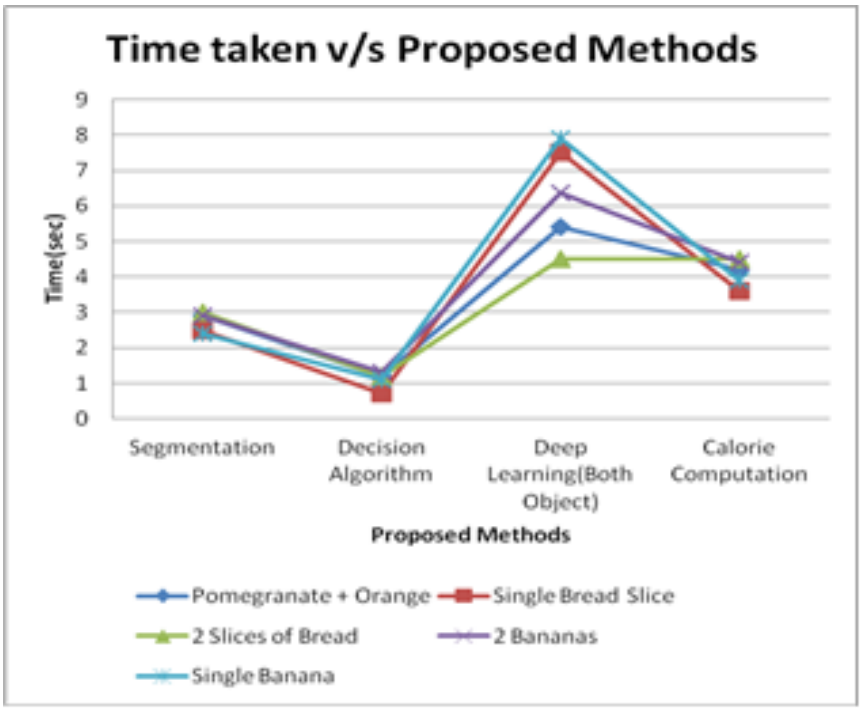


Figure 17: Comparison between time taken and proposed methods

6.4 Energy Consumption

The above-mentioned distributed cloud computing models, along with the decision algorithm, significantly reduce the overhead of the mobile device in terms of memory, CPU utilized, battery consumed, and storage capacity used. By calculating the consumption of these resources, we can compare how the cloud model has contributed to optimizing the use of these resources.

We were able to evaluate our e-health based mobile application, EHSH, based on the parameters of network usage, battery consumption, and CPU utilization, when the application was made to run on a local client server or on a cloud-based virtualization

server. Based on **Figure 18**, we can observe that the Android-based e-health application, when made to run on the local client server, consumes 744mW of battery usage of the mobile device, as compared to the 475 mW used when the application is made to run on the cloud virtualization model. The cloud-based virtualization model is able to completely take over the processing capacity of the application of the mobile device, which directly contributes to lower battery consumption (Figure 18).

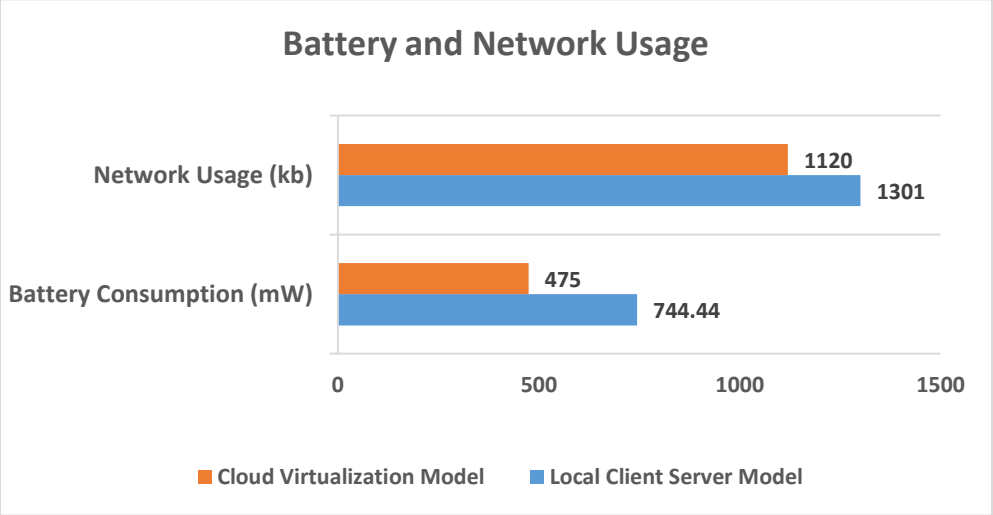


Figure 18: Comparison Based on CPU Utilization of the e-Health Application.

From **Figure 18**, we can also see that the network consumption of the Android-based e-health application (EHSH) is 1120kb when implemented based on the cloud virtualization model. This is far less than the 1301kb required to implement the application on the local client server model. There is an improvement of 181kb when the mobile application is made to run in the cloud-based virtualization environment.

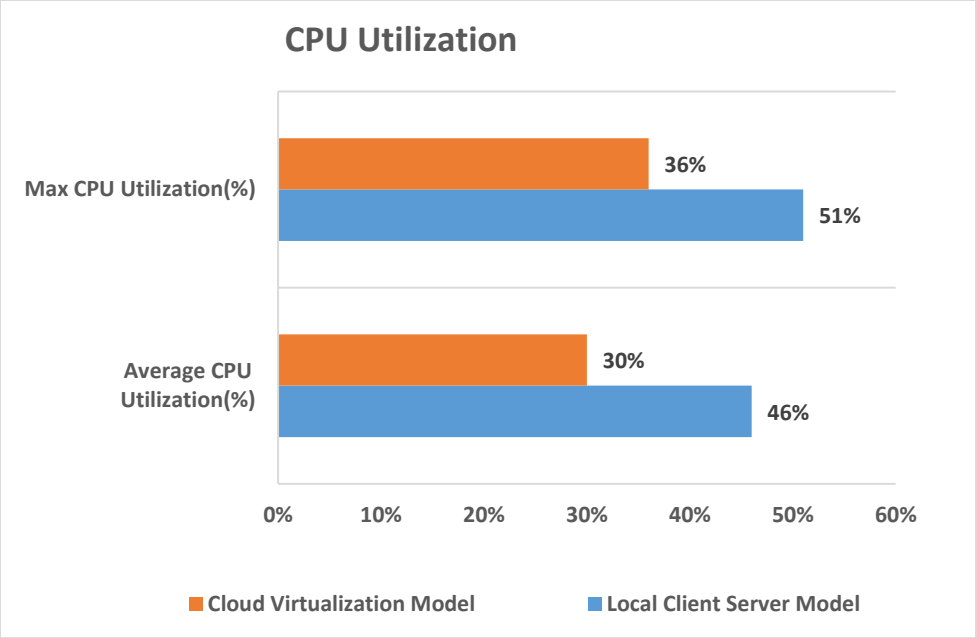


Figure 19: Comparison Based on Battery & Network Usage of the e-Health Application.

From **Figure 19**, we can observe that the EHSH application consumes 16% more CPU resources when the application is made to run on the local client server model, compared to the cloud-based virtualization model. Similar improvements to CPU utilization can be observed when an application is made to run in the cloud-based virtualization model, which results in an overall improvement of 15% (as shown in **Figure 19**).

Based on the results in Figure 15, Figure 16 and Figure 17 the overall time taken by the cloud based virtualization model was significantly less than the cloud based map reduce model by 155 sec. Also based on Figure 18 and Figure 19, with the use of the cloud based virtualization model, we noticed an improvement in terms of the resource consumption by the mobile device, in terms of the battery consumption and CPU Utilization. Hence the cloud based virtualization model is a preferred cloud based framework out of the two mentioned models.

Chapter 7

Conclusion

7.1 Discussion

We have discussed two major distributed cloud models and how they were integrated into a multimedia e-health application to achieve better accuracy and faster processing. The proposed distributed model, which implemented the support vector machine in parallel in the cloud with the help of the MapReduce technique, addresses the issues of scalability and parallelism inherent in a split data set. We implemented the distributed model in the cloud environment using cloud services from Amazon Web Services, through which we were able to run our SVM classification algorithm on multiple Amazon EC2 instances using the MapReduce methodology. This methodology creates master and slave instances, which implement the SVM algorithm across multiple worker instances. We evaluate this model on multiple cluster nodes (a three-node cluster, a four-node cluster, a five-node cluster, and a six-node cluster) to analyze the overall timings for each of these environments. We observe that the map and the reduce tasks contribute significantly to overall timings. Moreover, the analysis is based on the assumption that we do not take into account the initial time taken to allocate the instances or the bootstrapping time required by the Amazon EC2 instances. Even though this model implemented the distributed model and made use of cloud resources to offload processing from the mobile

device to the cloud, the overall time usage required significant improvement to reduce the response time of the mobile e-health application.

Although the MapReduce-based cloud model yielded the desired results for larger training sets, the initial time needed to distribute the datasets and process them across resizable Amazon EC2 instances was considerably higher for smaller training sets of data. Moreover, since the previous system was based on a client-server architecture model, we had to re-write the computer vision code for it to run in the cloud-based MapReduce environment.

As a result, we opted for a cloud virtualization methodology, wherein both the mobile and the cloud platform existed on the same operating system. Since both existed on the same platform, we did not have to rewrite the computer vision code; instead, the same code was functional on both the mobile and the cloud server. Following the hardware configuration changes necessary to achieve cloud virtualization, we were able to achieve faster and better results than in our previous model. Hence, we opted to use the virtualization methodology over the previous MapReduce task.

Cloud virtualization addresses the issues of scalability by creating a virtual image for each user. By enabling virtualization in our model, we are able to run our multimedia e-health mobile application in the cloud with the Android x86 image. Specifically, we describe a cloud-based virtualization mechanism for a multimedia-assisted mobile food recognition application. Our mechanism allows users to control their virtual smartphone operations through a dedicated client application installed on their smartphones, while the

processing of the application continues to run on the virtual mobile image, even if the user is disconnected for some reason. With our mechanism, the mobile environment is emulated on the cloud in such a manner that users always feel as if the application is being run on their smartphones (while, in reality, it is virtually running on the remote server in the cloud).

We also noticed an improvement in terms of the resource consumption by the mobile device, in terms of the battery consumption and CPU Utilization when adopting the cloud based virtualization model over the MapReduce model. Hence the cloud based virtualization model is a preferred cloud based framework out of the two mentioned models.

Overcoming the limited capability of smartphones to run intensive machine learning is a rather significant achievement. To accomplish this, we propose an approach in which the mobile session of our e-health application, EHSH, is swapped from the cloud session running the same mobile application to the mobile device. This approach will enable faster processing times for running machine learning algorithms (in our case, deep learning and calorie measurements) with limited resource usage.

7.2 Future Work

In future work, we would like to enhance the decision-making algorithm further by adding additional parameters, such as the ability to analyze previous scenarios in which multiple food objects were sent to various cloud servers. By keeping a record of such previous

scenarios, the system will able to make intelligent decisions regarding which cloud server to send images to in order to reduce total time usage.

The concept of creating multiple virtual images of an application, in cloud, is not only confined to e-health application and also not just confined to the mobile device. As part of the future application of this approach, this approach could be used for switching between the multiple copies of the same application across different computing devices like, desktop machine, mobile device, tablet device, laptop etc.

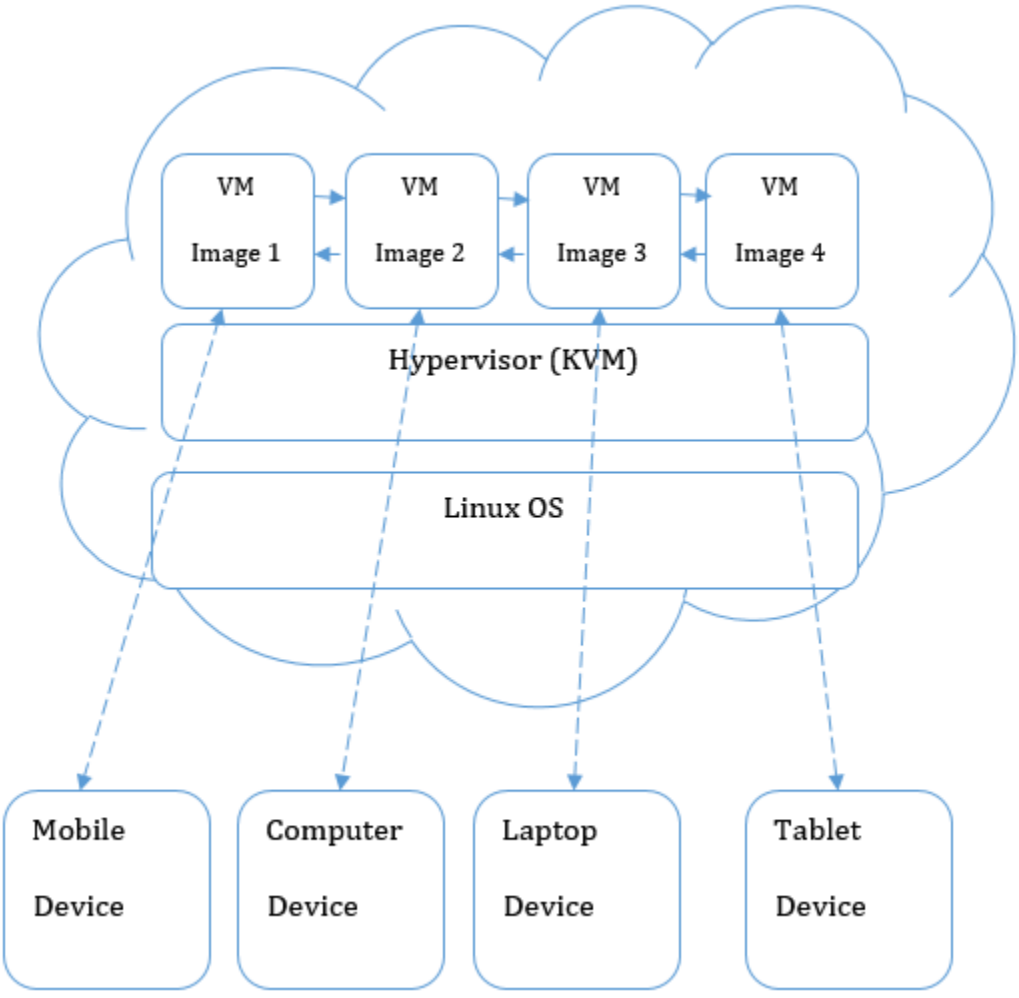


Figure 20: Session Transition between Multiple Virtual Images

As shown in Figure 20, the above mentioned approach could be used to switch between the virtual images across many devices. Since the virtual images would be running in cloud, the application could always remain in cloud, and the with replication setup across the virtual images, the data could remain consistent. Also each of the virtual image could be assigned each of the physical device, with the corresponding compatibility settings.

References

[1] Ford ES, Giles WH, Dietz WH: Prevalence of the metabolic syndrome among US adults: findings from the third National Health and Nutrition Examination Survey. JAMA 287:356–359, 2002.

[2] Kemp, Roelof, Nicholas Palmer, Thilo Kielmann, and Henri Bal. Cuckoo: a computation offloading framework for smartphones. In Mobile Computing, Applications, and Services, pp. 59-79. Springer Berlin Heidelberg, 2012.

[3] Marinelli, Eugene E. Hyrax: cloud computing on mobile devices using MapReduce. No. CMU-CS-09-164. CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2009.

[4] Pouladzadeh Parisa., Bharat Peddi Sri Vijay., Kuhad Pallavi., Yassine Abdulsalam., Shirmohammadi Shervin.: Mobile Cloud Based Food Calorie Measurement. IEEE International Confernce on Multimedia and Expo Workshops (ICMEW), pp 1-6, Chengdu, China (2014)

[5] Pouladzadeh P., Shirmohammadi S., Bakirov A., Bulut A., Yassine A.: Cloud-Based SVM for Food Categorization. Springer's Journal of Multimedia Tools and Applications, DOI 10.1007/s11042-014-2116-x, (2014)

[6] Pouladzadeh P., Shirmohammadi S., Yassine A.: Using Graph Cut Segmentation for Food Calorie Measurement. IEEE International Symposium on Medical Measurements and applications, pp 1-6, Lisbon, (2014). – 6

[7] Liu Xuan, Master's Thesis, University Of Ottawa (https://www.ruor.uottawa.ca/handle/10393/30702)

[8]Web Link: http://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html

[9]Website Link: http://www.amd.com/en-us/solutions/servers/virtualization

[10]Website Link: http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf

[11] Sobhy D., El-Sonbaty Y., Abou Elnasr M.: MedCloud Heathcare Cloud Computing System. IEEE Internet Technology and Secured Transactions, pp 161-166 (2012)

[12] Doan B.H. and Chen L. : Mobile Cloud for Assistive Healthcare. IEEE Asia-Pacific Serices Computing Conference APSCC, pp 325-332, (2010)

[13] Yong W.A, Cheng A.M.K., Baek J., Jo M., Chen H.: An Auto-scaling Mechinam for Virtual Resources to Support Mobile, Pervasive, Real-Time Healthcare Applications in Cloud Computing. IEEE Network, Volume 27, Issue 5, pp 62-68, (2013)

[14] Harini M., Bhairavi K., Gopicharan R., Kirupa G., Vaidehi V.: Virtualization of Healthcare Sensors in Cloud. IEEE International Conference on Recent Trends in Information Technology, pp 663-667, (2013)

[15] R. Van Nieuwpoort, J. Maassen, G. Wrzesińska, R. Hofman, C. Jacobs, T. Kielmann, H. Bal, “Ibis: a flexible and efficient java based grid programming environment”, Concurrency and Computation: Practice and Experience 17 (2005) 1079–1107

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

[17] Kosta, Sokol, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang: Thinkair: Dynamic resource allocation and parallel execution.

[18] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu, “Mobile cloud computing: A survey”, Future Generation Computer Systems , 2012.

[19] Kemp, R., Palmer, N., Kielmann, T., Seinstra, F., Drost, N., Maassen, J., Bal, H.E.: “EyeDentify : Multimedia Cyber Foraging from a Smartphone.” In: IEEE International Symposium on Multimedia (2009).

[20] D.Borthakur, “The hadoop distributed file system: architecture and design”, (http://hadoop.apache.org/common/docs/r0.18.0/hdfs_design.pdf), 2007.

[21] Sri Vijay Bharat Peddi, Pallavi Kuhad, Parisa Pouladzadeh, Shervin Shirmohammadi, “A Map Reduce Parallel Classifier for Cloud Based Food Recognition” in International Conference on Next Generation Computing and Communication Technologies [ICNGCCT]-2014, Dubai. pages 142-147.

[22] http://www.linux-kvm.org/page/Main_Page

[23] http://developer.android.com/tools/devices/emulator.html

[24] Parisa Pouladzadeh, Sri Vijay Bharat Peddi, Pallavi Kuhad, Shervin Shirmohammadi, "A Virtualization Mechanism for Real-Time Multimedia-Assisted Mobile Food Recognition Application in Cloud Computing" in Cluster Computing Journal 2015.

[25] Deng L, Yu D. Deep Learning: Methods and Applications, Foundations and Trends in Signal Processing. 2014;7: 197–387. http://dx.doi.org/10.1561/2000000039.

[26] http://neuralnetworksanddeeplearning.com/chap2.html.

[27] Krizhevsky, A., Sutskever, I., and Hinton, G. on "ImageNet classification with deep convolutional neural networks." Advances in Neural Information Processing Systems (NIPS), 2012in NIPS', 27 pages, 2012.

[28] N. Srivastava and R. Salakhutdinov, & ldquo, "Multimodal Learning with Deep Boltzmann Machines", Proc. Neural Information and Processing System, 2012.

[29] Pallavi Kuhad, Abdulsalam Yassine and Shervin Shirmohammadi, "Using Distance Estimation and Deep Learning to Simplify Calibration in Food Calorie Measurement" in Computational Intelligence and Virtual Environments for Measurement Systems and Applications [CIVEMSA]-2015, Shenzhen, China.

[30] Sun Z, Fox G. Study on Parallel SVM Based on MapReduce. In: International Conference on Parallel and Distributed Processing Techniques and Applications; 16-19 July 2012; Las Vegas, USA. CSREA Publishing. pp. 495-561.

[31] M.Bhonde, P.Patel. Efficient Text Classification Model Based on Improved Hypersphere Support Vector Machine with MapReduce and Hadoop in Volume 3, Issue 8, August 2013

ISSN: 2277 128X International Journal of Advanced Research in Computer Science and Software Engineering.

[32] http://www.android-x86.org/

[33] Peter Mika, Distributed indexing for semantic search, Proceedings of the 3rd International Semantic Search Workshop, p.1-4, April 26-26, 2010, Raleigh, North Carolina.

[34] Kanetkar, S.; Shahnasser, H., Virtual screen for cloud thin clients, Digital Information Processing and Communications (ICDIPC), 2012 Second International Conference on , vol., no., pp.102,107, 10-12 July 2012 doi: 10.1109/ICDIPC.2012.6257275.

[35] Chen, E.Y.; Itoh, M., "Virtual smartphone over IP," World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a , vol., no., pp.1,6, 14-17 June 2010 doi: 10.1109/WOWMOM.2010.5534992

[36] Rehunathan, D.; Bhatti, S.; Chandran, O.; Pan Hui, "vNurse: Using virtualisation on mobile phones for remote health monitoring," e-Health Networking Applications and Services (Healthcom), 2011 13th IEEE International Conference on , vol., no., pp.82,85, 13-15 June 2011.

[37] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Her-bert Bos. Paranoid android: Versatile protection for smartphones. In ACSAC , 2010.

[38] [5] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: Elastic execution between mobile device and cloud. In Proc. of EuroSys , 2011.