Hyrax: Crowdsourcing Mobile Devices to Develop Proximity-Based Mobile Clouds

Chye Liang Vincent Teo

CMU-CS-12-131

August 2012

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Priya Narasimhan, Chair
Daniel Siewiorek

Submitted in partial fulfillment of the requirements for the degree of Master of Science.

This research was sponsored by the National Science Foundation under grant number CNS-090754.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the US government or any other entity.

Keywords: mobile, cloud, grid, distributed, computing, Hadoop, Android, Hyrax, MapReduce, smartphones, middleware, filesystem, peer-to-peer

Abstract

The computation and storage capabilities of today's mobile devices are rapidly catching up with those of traditional desktop computers and servers. In fact, multi-core mobile phones with 1 GHz processors are readily available in the market today. Mobile devices also have more onboard resources, typically 512 MB of RAM or more available. Furthermore, tablet computers, which are even more resource-rich, are increasingly prevalent with projections of 195 million tablets to be sold by 2015. This implies that there are plenty of computing resources present within our vicinity, and literally in our hands, in our everyday lives.

Unfortunately, all these processing and computation resources are mostly under-utilized as mobile devices are generally used to process local data and programs only. In other words, devices mostly operate independently from each other. Any data that they share usually has to go through a central content server, which involves the use of global data networks (i.e., either a Wi-Fi connection or a 3G/4G cellular connection) to access the Internet in order to communicate with the central server. Any computation that these devices offload (perhaps to dedicated cloud-hosted services) also typically involve communications through the Internet. However, with an increasing number of both mobile and Internet users globally, the bandwidth of networks that form the Internet are getting strained. This means that users usually experience a perceptible delay before getting results back from the content servers they are communicating with through the Internet. Also, given the richness of resources that a collection of mobile devices can constitute, there might be alternative ways of exploiting those collective resources to provide benefits for the user.

To overcome these constraints, we propose to utilize local wireless networks to enable mobile devices that are within the vicinity of each other to communicate directly without utilizing either the resources of a global cellular network or the Internet. We believe that crowdsourcing the mobile-computing resources within a vicinity has the potential to enable collaborative data-intensive computing across a cloud of mobile devices within the same network without straining the bandwidth of global networks. Effectively, the collection of mobile devices that collaborate in such a manner represent a genuine mobile cloud. Such collaborative computing efforts could also potentially take advantage of the locality and/or data from sensors (such as GPS, temperature, etc.) that are prevalent in smartphones today. Such a system also has the advantage of speed because the communications only need to go through one hop to get to their respective destinations.

To achieve these objectives, we have designed and developed Hyrax, a MapReduce system derived from Hadoop that supports cloud computing on a networked collection of Android mobile devices. Mobile users will then be able to leverage the collective computational resources of devices in their vicinity through Hyrax to run distributed jobs on readily available data stored on their devices. We evaluate the resource usage (bandwidth, CPU, memory, battery usage, etc.) of using Hyrax on a collection of mobile devices for standard MapReduce benchmarks that are traditionally run for a cloud of desktop, rather than mobile, computers. We also develop and evaluate a music-sharing application to demonstrate an application of Hyrax in a possible real-world scenerio, where users can share, and make available for download, the music files stored on their mobile devices with other users in their vicinity.

Acknowledgments

Acknowledgements are due to the many people who have supported me, either directly or indirectly, in completing this work.

Firstly, I would like to acknowledge and thank Professor Priya Narasimhan, my thesis adviser, for all her help, guidance, advice and funding for the past two years that I have worked with her.

Acknowledgements are also due to the Defence Science and Technology Agency, Singapore, for sponsoring my education for the last 4 years. I would never have had the opportunity to attend Carnegie Mellon University if not for their scholarship.

I would also like to thank Tan Jiaqi for his support, encouragement and occasional input of ideas for this project over the last 2 years.

Thanks is also due to Deborah Cavlovich for all her administrative help with the Fifth Year Master's Programme.

My sincere gratitude to my friends Esther, Benjamin, Heow Hui, Ruthika, Hannah, Yaqi, Junkai, Allen, Yiling, and (especially) Wee Hong for their words of encouragement, patience, listening ears and company when I needed it. Special thanks is also due to my roommate, Hui Han, for bearing with me all these years and for his excellent culinary skills.

Finally, I would like to thank my parents, sister and brother-in-law for their support, care and concern over the past few years of my education away from home. They have been a constant source of care and comfort, even when I have not always shown my appreciation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Despite the advent of cloud computing and the increasing use of cloud services by mobile phone users, the resources of mobile devices today are still primarily only utilised locally. Given that the computational and storage capabilities of mobile devices today are rapidly catching up with those of our traditional desktop computers and servers, it seems viable to utilise these resources for collaborative data-intensive computing across a cloud of mobile devices. In fact, multi-core mobile phones are already available in the market. Unfortunately, these computational resources are currently underutilised as they are generally only used to process local data and programs.

We developed Hyrax to utilise these resources on a network of Android smartphones to provide a local cloud computing infrastructure largely made up of these mobile devices. Hyrax is a MapReduce system based on Hadoop, which is an open-source Java implementation of the MapReduce system and the Google File System. Hyrax, together with the hardware resources of the mobile devices, could form the basis of a *local mobile-cloud* that can be used to compute jobs. Here, we define a local mobile-cloud as a cloud computing infrastructure that does not make use of global networks (ie the Internet).

There are already many mobile applications that obtain information or data from multiple mobile phones and process the data before displaying the results to users. Examples include websites and services such as Flickr, YouTube and Dropbox. Even Facebook now increasingly utilises location information from mobile users. The main difference between Hyrax and such applications is that current applications perform the computation on a centralised remote server (ie users upload their data to the server where it is processed), while Hyrax aims to bring the storage and computation of data to the local level on the networked set of mobile devices, thereby bypassing the global network altogether.

Hyrax allows mobile users to offer their smartphones' computing and storage resources to other users and applications within the same network. This has the potential to enable a collection of smartphones to execute large-scale distributed applications at little extra monetary cost, since unlike traditional cloud-based infrastructure, no major new investment in servers and other equipment is required nor necessary. Each device acts as both a computation and storage node, able to support local computation on (potentially local) data as part of a larger distributed job.

The use of mobile devices for cloud computing offers some advantages over the use of traditional servers and machines. For example, the use of mobile devices opens the possibility of making use of local sensor and multimedia data without substantial network transfer of data, since the computation can be performed locally on the node with the data. This gives rise to more efficient data access patterns, as well as distributed maintenance of hardware.

1.1 Background

We provide some necessary background in current smartphone technology, cloud computing, MapReduce and Hadoop, as well as Android in this section.

1.1.1 Smartphone Technology

Smartphone technology has seen numerous advances in the last several years, both in terms of hardware and software. Advances in hardware have resulted in more memory, sensors and computational resources being packed within one mobile device. A corresponding advance in software has allowed numerous applications that make full sue of the sensors and Internet connectivity available to the user. Examples include image and video capturing, playing of music and films, as well as games (some of which include collaboration with other players over the Internet). The use of location sensors has also enabled applications such as GPS navigation, which once required the use of a specialised GPS navigational system. Tha ability to use location sensors as well as image and sound capturing also means that smartphones are capable of producing and storing local data.

Furthermore, advances in hardware such as cheaper memory and faster processors also mean that smartphones are now capable of storing amounts of data and performing computations that were once only possible on traditional desktop machines. This implies that many of us now carry the resources of a desktop machine around with use in our bags

and pockets constantly.

The use of smartphones has also become increasingly common and widespread. According to a report by telecommuncations equipment and services provider Ericsson, mobile broadband subscriptions globally have reached 1 billion by the end of 2011, and is projected to hit 5 billion by 2017 [24]. The same report stated that there were approximately 700 million smartphone subscriptions by the end of 2011, and it is predicted that this figure will increase to 3 billion by 2017.

On a similar trend, tablet computers have also seen widespread adoptation for use in recent years. Being larger in size, tablet computers usually have more computation, memory and power resources than a mobile phone. In fact, studies have shown that $14\ \%$ of mobile subscribers in the US now own a tablet device as well. This figure is predicted to rise, and similar trends are predicted for other markets in Europe and Asia as well [16]. This further serves to increase the amount of computation power that is available at our disposal regardless of our location.

On the other hand, despite all these advances made in smartphone technologies, some aspects of this development are still left wanting. While numerous advances have been made in processor speeds and memory capacities, the same cannot be said for battery technology. Mobile devices are still severely constrained by the limitations of their battery capacities. In fact, short battery lives still have a huge potential in destroying customer satisfaction with their mobile devices [37]. Hence, any discussion of applications for mobile devices must take power consumption into account.

1.1.2 Cloud Computing

Cloud computing, according to [7], refers to "both the applications delivered as services over the Internet and the hardware and systems software in the datacentres that provide those services." In other words, cloud computing allows users to execute huge computational jobs on hardware provided at datacentres for a low cost. The benefits are two-fold: users get to execute their jobs at low cost without having to invest in their own infrastructure, while service providers get to profit when their machines are constantly utilised by their customers. Without the need to invest in their own hardware, users do not have to worry about under-investing or over-investing in hardware, especially when they sometimes only need the hardware for a few jobs.

An important aspect of cloud computing is virtualisation, where users are given the illusion of infinite computational and memory resources that are available when needed. With the advent of cloud computing services such as Amazon Elastic Compute Cloud

(EC2) [2], users can now utilise the power of huge clusters of computing on demand and only pay for what they use for. This gives them the ability to rapidly develop products or expand their business at low cost.

1.1.3 Cloud Computing & Mobile Devices

The idea of "mobile cloud computing" is not new. However, most research into mobile cloud computing to date have concentrated on using mobile devices as interfaces to access cloud computing resources and services provided on traditional servers over the Internet. In other words, the mobile devices only act as the client, offloading computation to other machines off-site.

The primary motivation for accessing cloud services on mobile devices is power constraints. As mentioned in Sub-Section 1.1.1, battery capacities are still wanting in today's mobile devices. Numerous studies have been conducted to investigate the potential power savings involved in offloading computations off-site from mobile devices. This involves a study into the tradeoff between the energy consumption of performing the computations locally and that of the communications involved in offloading the computations (and associated data) off-site [34, 31]. In other cases, the motivation for offloading computation is the prohibitively large size of the computations involved in the application, which makes it unsuitable to be executed on a mobile hardware device (for an example, see [3]).

Our work differs significantly from the above-mentioned works in that we do not aim to offload computations from the mobile devices to off-site servers that reside somewhere else over a network or even the Internet. Instead, we intend to offload computations to mobile devices in the vicinity of the client device. To date, however, performing such computations on mobile devices have primarily been explored in the context of grid computing (for example, see [11]).

One of the closest works that is similar to ours is [20], which also aims to develop a MapReduce framework for a mobile platform. One major difference between that work and Hyrax is the lack of a distributed file system in that work. We provide further details about this and other related work in Chapter 2.

1.1.4 MapReduce and Hadoop

MapReduce [18] is a programming model initially developed by Google. It was designed to process large sets of data distributed across a cluster of commodity servers. It was specifically designed to be scalable and fault-tolerant in order to accomodate both an ex-

pansion in cluster size as well as equipment failure. It is usually used in conjunction with the Google File System (GFS) [25], which is a distributed file system.

Under a MapReduce framework, jobs are essentially split into two main distinct phases: a "map" phase and a "reduce" phase. During the "map" phase, users provide a "map" function that accepts key/value pairs as input parameters and returns intermediate key/value pairs (which need not be of the same type). The intermediate key/value pairs are then sorted by key value (during an intermediate phase known as "shuffle") before being used as input to a user-specified "reduce" functin. This function then produces the final output. Both the initial input and final output are read from and written to a distributed file system.

MapReduce is able to parallelise jobs as it is able to split the initial input by key value and call the "map" function on each such portion of the initial input data. In this way, individual "map" functions on different nodes can run at the same time on different parts of the initial input data. The same principle applies during the "reduce" phase of the job, where intermediate key/value pairs with similar keys are handled by the same instance of the "reduce" function. In this way, the "reduce" phase of the job can also be parallelised and run on multiple nodes simultaneously. This entire operation is manged by a master node, which decides how the input data is split and which nodes will handle which portion of the input data. In order to minimise data transfers across nodes (and hence the job execution time), the master node will attempt to ensure that each node will work on data that is already stored on its local filesystem. If a data transfer between nodes is required, the master node will inform the slave node concerned and the transfer takes place directly between the two nodes without having to go through the master node. Readers might refer to [18] for a more detailed explanation of the MapReduce framework.

Hadoop [5] is an open-source implementation of the MapReduce model and is usually used in conjunction with the Hadoop Distributed Filesystem (HDFS) [39], which is itself based on the GFS. Hadoop is implemented in Java.

An instance of a Hadoop cluster consists of four main processes: NameNode, JobTracker, DataNode and TaskTracker. There is only one NameNode and one JobTracker running per Hadoop cluster, and these two processes are typically executed on one or two main master nodes. There is one DataNode and one TaskTracker instance running on each slave node of the cluster. The NameNode is responsible for maintaining the directory structure of the HDFS, and receives information about each block of data residing in each slave node from the individual DataNodes. The NameNode is also responsible for deciding where new data is to reside in the cluster. The JobTracker, on the other hand, is responsible for managing jobs and coordinates the execution of jobs and sub-tasks among the individual TaskTrackers. The TaskTrackers are responsible for executing tasks assigned to its corresponding node by the JobTracker, while the DataNaodes are responsible

for fetching and writing data blocks on its local node.

One main draw of Hadoop is its ability to tolerate faults. Hadoop (and MapReduce) was designed under the assumption that hardware faults will occur. Hadoop achieves fault tolerance by re-executing failed tasks on other nodes, and also by replicating each block of data among several DataNodes. Furthermore, Hadoop has the capability to redundantly execute multiple instances of the same task on different nodes (known as speculative execution) in order to increase the probability of success and decrease execution time. This capability to tolerate faults, as well as the ability to execute largely independent tasks and directly transfer data between slave nodes, makes Hadoop easily scalable to a cluster of a large number of machines.

Writing applications for Hadoop is not complicated as programmers typically are only required to specify two main functions: the "map" and "reduce" functions. The details of data storage, as well as data splitting and task distribution, are handled automatically by the Hadoop system and hidden away from the programmer. This takes a lot of the work away from the programmer, enabling rapid program development.

1.1.5 Android

Android [1] is an open-source mobile operating system developed by both Google and the Open Handset Alliance. Along with the Applie iPhone and Blackberry, Android is one of the more popular mobile operating systems in use today. It is built on the Linux kernel, and uses the Dalvik Virtual Machine to execute applications.

Dalvik was specifically designed to optimise applications for the limited computation, power and memory resources of a mobile device. Android code (written in Java) must be compiled to Dalvik bytecode (which have the `.dex` format) and packaged in a `.apk` file to be installed on an Android device.

1.2 Contributions

We aim to develop a mobile cloud-computing infrastructure mostly made up of mobile nodes. We present our implementation and evaluation of a mobile-cloud computing infrastructure based on MapReduce and Hadoop in this paper.

We focused on Hadoop since it was based on MapReduce, which was designed for data-intensive cloud computing on commodity clusters. Furthermore, Hadoop enjoys support among the industry, being used by companies such as Yahoo!, Facebook and IBM to

process large sets of data.

Hadoop was a suitable choice as it already includes the features and functionalities required for a mobile-cloud computing infrastructure, including the ability to handle node departures. The Android platform was chosen since it is based on the Dalvik Java VM, which allows us to execute much of Hadoop's original codebase without heavy modifications.

This project is a continuation of an initial version of Hyrax, and represents some improvements over the previous project by Marinelli [33]. While Marinelli's project was suitable for initial use to discover the resource constraints / challenges, performance and the scalability aspects of using mobile devices for collaborative data-intensive computing, we found that his initial implementation of Hyrax was not suitable for wide-scale deployment on the mobile devices of common users.

To that end, we have improved on Marinelli's implementation of Hyrax

This paper is organised as follows: Chapter 2 discusses some related work. We present our problem statement, goals and high-level approach in Chapter 3. Chapter 4 describes our implementation of Hyrax. Our empirical evaluation of Hyrax is given in Chapter 5. We provide an example of a use-case for Hyrax and our evaluation of it in Chapter 6. Finally, we discuss lessons learnt, possible future work, as well as present our conclusion in Chapter 7.

Note that all references to the "previous implementation" or "previous report" refer to the work described in [33].

Chapter 2

Related Work

Bahl et al [8] provide a rather comprehensive view of mobile cloud computing research and future directions. According to them, the current research on mobile cloud computing can broadly be categorised into two categories: using mobile devices as thin clients to access cloud services online, and using mobile devices as computational and storage nodes as part of a cloud computing infrastructure. These two models of mobile cloud computing are referred to as the "agent-client scheme" and "collaborated scheme" respectively by [29]. Another category is a hybrid of the previous two, where part of the computation is performed on the mobile client device itself, while the rest of the computation is offloaded to a cloud service. This is described as "augmented smartphone applications" by [15]. [29] also argue for the case of a middle-tier (ie "cloudlet"), which is a cloud computing service that is located in the vicinity of a mobile user so that clients can access cloud computing services directly without the use of the Internet. The use of "cloudlets" will greatly improve latency, which is important in many mobile applications where user interaction is involved.

Another category of work that is not directly related to mobile cloud computing is that of using mobile devices as nodes in a mobile sensor network. We feel that such work are interesting as the full potential of the mobile devices are not utilised in such cases, as they could also be used as computational nodes. Extending these works could result in interesting applications.

We provide an overview of the different categories of mobile cloud computing in the remaining sections of this chapter.

2.1 Mobile Devices as Thin Clients

Most research into mobile cloud computing assume a model where the mobile devices are no more than thin clients, offloading computation to other cloud resources on the Internet.

One prime example of such a work is that done by Satyanarayanan et al [38]. Instead of accessing cloud computing facilities over the Internet, they propose setting up small clusters of cloud computing resources in local areas such as cafes that mobile users can access through the use of wireless LANs. They term such clusters "cloudlets." Through the use of cloudlets, mobile users retain the advantage of offloading huge computation jobs to machines with more appropriate resources to handle them. An additional advantage is that due to the proximity of the cloud computing facilities, users no longer need to depend on Internet connectivity to access cloud services. Instead, network latency can be greatly reduced since only a one-hop latency is involved. Using a local network also provides greater bandwidth than a mobile Internet connection, which allows for faster transfer of data and code. However, they propose offloading entire virtual machines to the cloudlets in order to excute jobs. This entails the transfer of huge amounts of data as virtual machines are huge in size. The sizes of the virtual machines, even when compressed, were at least 100 MB. This involves a huge amount of data transfer before running jobs on the cloudlets. We describe an extention of this work in Section 2.2, which mitigates this problem.

Other works involved offloading computation to a cloud computing provider over the Internet. An example is [3], which is an application that makes use of the availability of sensors on mobile devices as well as leveraging the power of cloud computing resources on the Internet to detect traffic lights for the blind, where pictures taken by cameras on the devices are sent over the Internet to cloud computing clusters to be processed in order to detect the position and state of traffic lights at road junctions.

2.2 Mobile Devices as Computational and Storage Nodes

There has also been some research into enabling mobile devices to act as computational and storage nodes in a cloud computing infrastructure. These works are closer to what we are trying to achieve with Hyrax.

One such work extends that of [38] by extending the concept of cloudlets to include the use of mobile devices that are also in the vicinity. Verbelen et al [40] propose the use of available mobile devices in the vicinity of a user instead of a fixed infrastructure to provide local cloud computing services. Furthermore, they propose breaking down applications into components and offloading componenets instead of entire virtual machines. This not

only reduces the data transfer necessary to execute a distributed job, but also allows greater control over how the application is divided among the available resources.

2.2.1 MapReduce on Mobile Devices

Elespuru et al [23] developed a MapReduce system for mobile devices in order to explore the feasibility of deploying such a system on such devices. Their work was developed for the iPhone platform. Their results show that implementing a MapReduce system for mobile devices is feasible. Furthermore, they have some ideas for encouraging user participation in such a system, which includes providing incentives such as rewards.

The work which we find to be the closest to Hyrax is a system by Dou et al known as Misco [20]. They implemented a MapReduce system from scratch using Python, targeting the Nokia N95 smartphone. One major difference between Misco and Hyrax is that Misco does not contain a distributed file system. As a result, input and output data are stored on the server of their system. This makes the loss of the server even more critical than in a typical Hadoop system, as the loss of the server also means the loss of data from the system. This also increases data transfer when executing jobs as all mobile nodes must download the input data from the main server before task exeuction can begin. There are no opportunities for tasks to be assigned to nodes that already store the required input data.

2.3 Augmented Mobile Applications

Chun et al [15] introduced the idea of augmented mobile applications, where part of the application is executed on the local mobile device, while other parts (for example, computationally intensive tasks such as speech recognition) are offloaded to cloud computing infrastructure. They proposed a system called "CloneCloud", where the cloud computing infrastructure hosts clones of the actual mobile devices. They envisioned the development of (possibly annotated) mobile applications which could be analysed both statically and dynamically to determine which portions or blocks of the application could be offloaded to the cloud infrastructure.

[14] extends the work from [15] by actually implementing a "CloneCloud" system. Their system operates at the thread granularity, which means that they offload work to the cloud infrastructure at the thread level. Applications are partitioned offline through both static and dynamic analysis. The partitioning process is automated and requires no modification to the original application nor any programmer annotations within the application

code. Their results indicate that speedups of up to 20 times are possible when mobile applications are augmented as compared with executing them solely on mobile devices. Furthermore, such speedups are obtained without requiring any effort by developers to modify nor annotate their applications.

[26] is also a work based on partitioning an application (not necessary a mobile application) to run on both mobile devices and traditional servers. Applications are first represented by a data flow graph and the partitioning algorithm finds an optimal cut on this graph that maximises some objective function. Interestingly, the primary motivation for this work does not appear to be for mobile applications. Instead, they focus on making non-mobile applications accessible from mobile devices. In other words, the mobile device acts as an interface to the application. For example, one of the example applications they studied was a 3D home modelling application that was not originally designed for mobile users.

2.4 Mobile Nodes in a Sensor Network

Craciunas et al [17] present a rather novel concept of mobile cloud computing. They propose to use physical mobile nodes (aerial vehicles in this case) as platforms to host virtual sensor networks. These virtual mobile sensors, known as "virtual vehicles", have the ability to migrate from one physical vehicle to another depending on the operational need of their mission as well as the physical location of the actual vehicles. The entire network of physical vehicles and virtual vehicles is managed by a central server (the "virtual vehicle network monitor"), similar to how a NameNode and JobTracker manages the entire cluster in a Hadoop system.

[17] is interesting as it allows tasks (the virtual vehicles), which collectively make up a distributed job, to migrate from one hardware to another in the midst of execution in order to achieve better performance. This is in contrast to traditional cloud computing and mobile cloud computing, where tasks are typically executed to completion on a single machine. It opens up many possible applications such as environment monitoring. It could be inexpensive if deployed on vehicles and aircraft that are already in service on the roads and in the skies as no investment in running and maintaining dedicated vehicles is required.

Another work that makes use of mobile devices as part of a sensor network is [10], which makes use of sensors on mobile devices to determine events of interests in a social setting and record them. The mobile devices mainly only play the role of sensors, while the computation required to determine event of interests and create interesting videos of them fall on backend servers.

Chapter 3

Problem Statement and Motivation

We state our assumptions, requirements, goals, non-goals, motivation, approach and problem statement in this chapter.

3.1 Assumptions

This project makes certain assumptions about the available hardware, as well as the possible applications of Hyrax. A brief justification is given with each of the following assumptions:

- Hyrax will mostly be used to process data that are already available on the mobile devices. This is in contrast to the usual cloud computing infrastructure, where data for generic distributed jobs must be specially uploaded to the cluster or obtained from the Internet. We do not target such generic jobs as we feel that they derive little or no benefits from a mobile cluster as there are many other cost-effective solutions using machines that are more capable than mobile devices.

- A central machine which acts as the master node exists in the network and is able to communicate with all the mobile nodes. This is a requirement of our current implementation as we do not intend to port the code for NameNode and JobTracker over to run on mobile devices. We feel that this requirement is realistic as the master node must have a stable connection to the network. This precludes having a mobile node (or mobile nodes) act as the master as its connection to the network cannot be assumed to be reliable.

- The master node will not fail. This is an assumption inherited from Hadoop. The master node is a single point of failure for the entire cluster. While there is work being done to implement a secondary master node in Hadoop, we do not implement it for Hyrax. While such a feature could possibly be implemented in future, it is not necessary to how the viability of Hyrax. We therefore do not plan to recover from any master node failure.
- Data uploaded to the mobile cluster will not be modified (or will rarely be modified). This is actually an assumption inherited from Hadoop. We feel that this assumption is reasonable in the context of our mobile cloud infrastructure as well since we do not expect data generated by mobile devices (sensor logs, pictures, videos etc) to change after they have been produced.
- Users have an incentive to sacrifice some of their devices' resources in order to use Hyrax. We assume that users have a compelling reason to trade some of their devices' resources in order to derive some benefit from Hyrax-based applications. While we do provide some motivating examples in Section 3.8, this paper is primarily concerned with the technical aspects of Hyrax. Nevertheless, we recognise that it is important that such incentives exist in order for Hyrax to be utilised to its full potential.

3.2 Requirements

We now list the requirements of a mobile cloud computing platform, which must be able to serve the needs of any applications written for it.

- Users and applications should be able to access any data (with the appropriate permissions) that is stored in the cluster regardless of the actual physical location of the data blocks.
- The platform should be capable of splitting a job that uses data on the cluster's filesystem as input among many different nodes, with each node working on one portion of the total input data independently. The platform should then be able to combine the results from each node and present the final output to the client.
- The platform should be fault-tolerant. It should also be able to tolerate node departure and arrival without any adverse effects on jobs that are executing nor loss of data. This is important as we anticpate that node departures will occur frequently with mobile nodes due to their unreliable network links.

- The platform should be able to work on any mobile hardware that supports the application. Currently, this only includes mobile devices that run on the Android operating system. As there are many different models of mobile devices on the market that support the Android operating system, they must all be able to work together within the cluster. The same requirement applies to the master node as well.

- The platform should be able to scale with the size of the cluster. An increase in the size of the cluster should not have any negative effects on the latencies of any executing jobs.

Resource Usage

- The platform should not consume an excessive amount of battery power when running. This is an important requirement as battery power is a very limited resource on a mobile device. Users will not want to be part of a system if it consumes too much power and depletes the batteries too fast, rendering the device unusable.

- The platform should not require a large network bandwidth. There are two reasons behind this requirement: network connections on mobile devices are known to be unreliable and slow, and network usage accounts for a significant portion of a device's power consumption. In order to reduce network data transfers, the platform should, as far as possible, process data that resides on the device itself. Besides, excessive network usage can result in a decreased availability of services, depending on the number of nodes and users on the network.

- The platform should not require too much memory nor consume too much processor resources. Besides consuming power, using too much memory or too much of the processor cycles will affect the performance of other running applications on the device. In the case of the Android operating system, memory use is limited to 16 MB per application.

- The platform should be able to complete any given jobs in a reasonable amount of time. This is important for a mobile cloud computing platform, especially if the clients are the users of the devices. Clients are unlikely to have the patience to wait if their jobs take a long time to complete.

3.3 The Choice of Hadoop

In order to implement a mobile cloud computing platform, there is a choice between building such a system from scratch (as some of the previous work mentioned in Chapter 2 have done), or to modify an existing cloud computing platform. We chose to implement our mobile cloud computing platform based on Hadoop because Hadoop already meets some of the requirements listed above. It therefore seemed to serve as a good starting point. The fact that Hadoop was developed in Java also tied in well with the fact that Android applications are developed in Java as well. We explore some the advantages and disadvantages of this decision below.

3.3.1 Advantages

- Hadoop is able to support global data access through the HDFS, which is a distributed filesystem. Furthermore, data is transferred directly from one node to another by the HDFS, which saves on network bandwidth and data transfer latency.

- Hadoop supports distributed data processing through MapReduce. MapReduce divides jobs into independent tasks and assigns each task to a node in the cluster. This assignment considers the location of the physical data blocks, such that as far as possible, computation is performed by a node that already has the required data blocks locally in order to avoid excessive data transfers within the cluster.

- Hadoop was designed with the possibility of hardware failures. It was therefore designed to be fault-tolerant. Hadoop handles this by replicating the data stored on the HDFS among different nodes, as well as re-executing any failed tasks (possibly due to node departure or equipment failure). This tolerance for faults is exactly what is required for a mobile cloud computing platform, where the risk of equipment failure (include battery exhaustion) and node departure is higher due to the mobile nature of the hardware.

- Hadoop is designed to run on different types of hardware and machines within the same cluster. This is due to the fact that Hadoop abstracts things such as the communications between nodes or disk I/O so that they are not hardware dependent. This ensures that different hardware can work together on the same cluster running the same code.

3.3.2 Disadvantages

- Due to the fact that Hadoop was initially designed for traditional machines and servers, excessive CPU and memory usage were not of critical concern during the development of Hadoop. This presents a problem on mobile platforms, since CPU, memory and power resources are limited on mobile devices. See [32] for a more in-depth discussion about the energy consumption of Hadoop.

- The master node is a single point of failure in the cluster. If the master node fails, the remaining nodes in the cluster will have no knowledge about the file paths nor block locations of the data blocks that are already stored in the HDFS. Only the NameNode has this information. It will be impossible for any node to get any data from the HDFS, even if the data is stored locally. Similarly, if the JobTracker fails, no new jobs nor tasks can be assigned to the slave nodes. The failure of the master node will effectively render the entire cluster useless. For a method to mitigate the failure of the NameNode, see [35].

- Hadoop makes extensive use of technologies such as XML for its configuration files. XML is, however, computationally expensive to parse and process. This is all the more evident on mobile devices. Further inefficiency is exposed through the use of servlets, which incur significant overheads, to make intermediate results available during the course of excution of a job.

3.4 Hadoop's Assumptions & Mobile Cloud Computing

As briefly mentioned in Section 3.1, some of the assumptions made by Hadoop fit in well within the constraints of a mobile cloud computing platform. These assumptions include:

- Hadoop's assumption that hardware will fail at some point is especially applicable for mobile devices. In the context of mobile devices, a disconnection from the network is also equivalent to a failure of equipment. Network disconnections are frequent for mobile devices. These can be due to signa losses, equipment failure, battery exhaustion or simply a result of power-saving mechanisms.

- Hadoop's assumption that it is easier to move the computation (ie code) than to move the data fits in really will within the context of a mobile cluster. Moving the code instead of data significantly reduces the need for network I/O since code is usually

much smaller in size than data. It therefore makes perfect sense to move the code instead of the data.

- Hadoop assumes that files are not modified after they are created / written. This ties in well with our assumption that Hyrax will mainly be used on data that originate from the nodes of the cluster itself (such as sensor logs, photos, videos etc). Such data typically do not change once they are written.

On the other hand, some of Hadoop's assumptions do not fit in well within the context of a mobile cloud computing platform. These include:

- Hadoop assumes that any applications written for it will make use of large datasets. It therefore sets the default block size to 64 MB, which is quite substantial. This size, however, does not correspond to the typical filesizes of sensor logs or photos or any other types of data that are typically stored on mobile devices. Using a blocksize of 64 MB will result in a lot of wasted disk space. One possible workaround would be to combine numerous files into one single file stored on a single block. Another (much simpler) workaround would be to simply reduce the blocksize.

- Hadoop and HDFS assume that jobs will not be interactive and therefore some degree of latency is tolerable. Consqeuently, HDFS is designed for batch-processing. This assumption does not correspond with the uses of a mobile applications, which typically involves real-time interaction with a user who expects the results of his or her request to be delivered within a reasonable amount of time, if not immediately. Hadoop is, in this respect, unsuitable for use on a mobile cloud computing platform targeted at mobile users. Readers may refer to [12] for some ideas on making Hadoop more suitable for interactive use.

3.5 Hadoop on Android

It was an obvious choice to port Hadoop over to run on the Android operating system. This was mainly due to the compatibilities between many of the libraries provided by the Android and the standard Java libraries used by Hadoop. Most of the code of Hadoop was therefore able to run on Android without any modifications.

Furthermore, it is easy to develop applications for the Android operating system as its SDK is easily available for download and the operating system allows any application to be installed on its devices. This is in stark contrast to the iPhone, another popular mobile

operating system on the market. In order to be able to install an application on the iPhone, developers will first need to have a developer account, which is expensive to purchase. Android's relative open-ness as compared to other mobile operating systems makes it an attractive option as the platform for experimentation as it eases the development process.

3.6 Problem statement

This project aims to explore the issues related to developing a mobile cloud computing infrastructure. The infrastructure should allow users and applications to utilise the resources of a mobile devices to perform collaborative computation tasks without affecting the mobility of the device itself. Furthermore, since cloud computing platforms for traditional clusters are already widely available, it makes sense to try to adopt them for use on a mobile cloud computing platform. Specifically, we aim to address the following issues:

1. What are the different requirements between a traditional cloud computing infrastructure and a mobile cloud computing infrastructure? What modifications to a existing cloud computing platform must be made in order to adopt it for use on a mobile cloud computing infrastructure?

2. How does the limitations in terms of power, storage and computational resources affect the effectiveness of a cloud computing platform? Is performance serverely affected?

3. What issues are involved when porting a cloud computing platform over to support a mobile cluster? How much work is involved in resolving these issues? Would it be better to build a new platform from scratch in order to support a mobile platform?

4. What possible applications are there for such a mobile cloud computing platform? Are there any practical applications that would benefit greatly through the use of such a platform?

This thesis will make the following claim:

Thesis Statement: *A mobile cloud computing infrastructure can be built through the adaptation of existing cloud computing platforms to provide local cloud services through the use of local data and computational resources.*

3.7 Goals

The goals of this project are as follows:

- Provide motivations for the use of mobile devices in a cloud computing architecture by citing advantages of such a platform, as well as how that it is feasible for such a platform to be developed.
- Implement a cloud computing platform for a mobile cluster. This shall be done through adapting the code of Hadoop for use on the Android platform for mobile devices.
- Evaluate the performance of Hyrax, including any improvements over the previous version of Hyrax, and any effects on Hadoop due to the use of a moblile platform instead of a traditional static cluster.
- Implement an application on top of Hyrax and evaluate its performance.

We do *not* aim to do the following:

- Develop a system / infrastructure that is fully ready for real-world deployment. This project is still at a proof-of-concept stage, and will be some way off from a real-world deployment. For example, our implementation only has support for local WiFi, but a real-world deployment implementation could possible support other forms of communications such as 3G mobile networks.
- Develop a system that is intended to completely replace a traditional cloud computing static cluster. We aim for this infrastructure to mainly support distributed mobile applications, and not take over many large-scale distributed applications that are supported by clusters today.

3.8 Motivation

The primary motivation for developing a mobile cloud-computing infrastructure is to better utilise the processing capabilities of a mobile device, which sit idle most of the time. Furthermore, the use of mobile devices as nodes in a cloud-computing cluster offers some advantages over static machines. Examples include the use of sensor and location data as input to distributed applications. Furthermore, advances made in mobile devices technology has made the use of mobile devices in such a platform feasible.

3.8.1 Advantages of Mobile Devices

Cloud computing systems are usually built on clusters of static servers. Large amounts of data are uploaded onto these servers and computations are performed on them. The data usually originates from external sources, and has to be specially uploaded to the system. In contrast, we envision that the data to be used in a mobile cloud computing system will mostly originate from within the cluster itself, specifically the mobile nodes. Since the data originates from the nodes, it can be processed locally as well, which reduces the frequency of data transfer over the network.

However, the use of mobile devices over traditional servers obviously posts some challenges as well. The first challenge that comes to mind is the limited resources, in terms of power, computation and storage, available on mobile devices. Furthermore, the network connectivity of mobile devices is far less reliable than that of a traditional static clusters. Despite all these challenges, the use of mobile devices still offers some advantages over static servers for a cloud computing infrastructure:

- There is no need to specially upload data to the cluster as the data is already available since it was generated from within the cluster. The data can also be processed locally by the node which generated it. This eliminates the need to transfer the data before processing can take place, saving precious network bandwidth.

- Since there is no need to transfer the data in from external sources, the data can easily be shared among other nodes through direct peer-to-peer connections within the cluster, which is both more efficient and cost effective. It also removes dependence on the global network connection as the service of an external centralised server is not required.

- It will be easier and more cost-effective to maintain the cluster as the ownership of the hardware (ie mobile devices) is distributed among the mobile users. These users will ensure that their individual devices are always working, since they use their devices for their personal needs as well.

- More and more mobile devices in use today have the capabilities to perform computation like any traditional machine. Besides, as the use of mobile devices is already widespread, the hardware is already available. A mobile cloud infrastructure could therefore potential consist of more machines than a traditional cluster at only a fraction of the cost.

3.8.2 Possible Applications

In this section, we explore some possible applications that could be implemented on top of a mobile cloud computing system. Such applications could possibly make extensive use of data obtained through the sensors and logs of the individual mobile nodes. Computation could be performed on the data directly and the results returned.

We explore the use of Hyrax for a music search and share application in Chapter 6. This idea can be further extended to searching for a sharing other types of media that can be found on mobile devices, typically photos and videos. Other possible applications include making use of the sensors on the devices to provide information such as traffic speed on the highways, crowd distribution within a building or area (such information could be useful, for example, during a fire evacuation) or temperature monitoring within a building in order to adjust the thermostat.

One compelling example of an application for Hyrax is searching for a particular feature in images. For example, imagine an event (such as university commencement) where hundreds and thousands of attendees are snapping photos using their mobile phones. If a person wanted to get all the images with his or her face in it, or images capturing a certain event, he or she would have to contact as many attendees as possible and request that they look through their pictures manually and send back the relevant ones. This is not only a time-consuming process, but almost impossible to achieve. With Hyrax, however, one could simply upload a sample image (for example, his or her face), which would then be distributed among all the mobile nodes. Each individual node could then perform a search on the photos that were taken by their owners locally and return only the relevant ones. Not only is this convenient for the user, but it is also much more efficient than the previous scenario. Hyrax actually makes such a huge task feasible.

3.8.3 Feasibility

We feel that the use of mobile devices as the hardware for a cloud computing infrastructure is technically feasible given the advances made in terms of both hardware and software. Besides CPU and RAM resources, the networking capabilities of mobile devices are also diverse now. Networking options including WiFi, 3G (or even 4G) mobile networks as well as Bluetooth are approaching speeds that are more than capable of supporting cloud computing traffic. For example, the Samsung Nexus S (one of the phones we used during the course of developing Hyrax) has a 1 GHz processor with 512 MB of RAM available, as well as 16 GB of internal storage. It has the capabality to use WiFi, 3G and Bluetooth to communicate with networks or other devices.

Furthermore, mobile devices now run operating systems that are based on those of desktop machines such as Linux or Mac OS. The most popular mobile operating systems include Android and iOS. Developers are able to write applications for these operating systems through the SDKs provided for these operating systems, much like writing programs for traditional operating systems running on desktop machines and servers. Furthermore, mobile applications are usually written in a programming language that is derived from one that is already widely used. For example, applications for Android are written mostly in Java, while those for iOS are written largely in Objective-C and C. This makes it feasible to port code that was already written in these languages to run on the mobile operating systems.

The main obstacle in terms of hardware for a mobile cloud computing infrastructure is the power resources of mobile device. Unfortunately, advances in battery capacities have not kept pace with advances in processor speeds or memory densities. This is a serious concern as running a cloud computing application will require power resources, and the cloud computing application will be competing for this resource with the other applications running on the device. It is important that power consumption is kept to a minimum.

3.8.4 Cost-Benefit Tradeoff

Given that users are required to sacrifice some of their devices' resources (disk space, network bandwidth, CPU, battery life etc) to be part of a Hyrax cluster, there must be some incentives and benefits involved for users before they will be willing to participate in a Hyrax cluster. We have to explore what makes using Hyrax beneficial to users such that it outweighs the cost of doing so.

One possible measure would be the job execution time. If a job can be completed in a significantly shorter amount of time with Hyrax than running it on a single device, it might be enough to convince users who want to execute similar jobs to participate in a Hyrax cluster.

Assuming that preparing and setting up a Hyrax job takes time s, ideally, we should have that the running time of a hyrax job on a cluster of n nodes is

$$t' = \frac{t}{n} + s$$

,

where t is the time taken to run the job on one single node alone (without Hyrax). In

order to achieve this, we need $s < t\left(\frac{n-1}{n}\right)$. This is easier to achieve with a higher value of n. Unfortunately, it is difficult to quantify the tolerance of users in terms of the need to sacrifice their devices' resources.

Another cost-benefit tradeoff that needs to be considered is the block replication factor. This factor determines how many copies of each block of data is stored within the cluster. This factor is a tradeoff between the amount of disk space required of users to store the replicas of each block, and the data availability of the cluster. A further discussion of block replication is given in Section 4.3.

Chapter 4

Implementation

In this section, we describe how Hadoop was ported to the Android platform, and we also provide an overview of the issues we encountered in the process of doing so. We found that the main challenge when porting the Hadoop code lay in the differences between the Android API and that of standard Java, which are not completely compatible.

4.1 Architecture

The architecture of Hyrax is shown in Figure 4.1.

From Figure 4.1, we can see that only the master node is connected to the router by wired ethernet. The mobile nodes are connected to the router by wireless LAN, and all communications within the network go through the router.

The master node hosts both the NameNode and the JobTracker. The NameNode obtains information from each DataNode in order to determine what blocks reside on each DataNode. It is also responsible for determining where nodes should get data from, and which nodes should data be written to. The JobTracker, on the other hand, determines how to distribute tasks among the TaskTrackers, and is responsible for coordinating the execution of tasks for each job. The master node is shown in Figure 4.3.

Each mobile node in the cluster runs an instance of both DataNode and TaskTracker and acts as a slave node. Disk space (typically in storage such as MicroSD) is also set aside on each mobile node to host blocks of data for the HDFS. The DataNode manages this storage space. Furthermore, the TaskTracker manages the computation tasks that are undertaken by this node. A detailed description of each mobile node is shown in Figure

Figure 4.1: Basic architecture of Hyrax

4.2. Mobile nodes are where the concrete storage of data and actual computation takes place within the cluster. Mobile nodes also have the potential to act as clients that send jobs to the cluster. Users do this through the client application installed in each mobile node. The client applications will then interact with the cluster through the NameNode and JobTracker on the master node via the "cloud".

It is extremely important to note that the "cloud" shown in Figure 4.1 exists logically, but not physically. Physically, the storage space and computation capabilities of the "cloud" are located within each worker node (which are the mobile nodes), as shown in 4.2. However, these details are hidden from the client, and only the "cloud" is presented as a virtualised interface to the client to access all these services within the cluster.

We do not assume that any of the devices nor the router are connected to the Internet, as we envision that Hyrax will depend only on data that is available within the devices in the cluster. This is, after all, one of the motivation for developing Hyrax: to reduce reliance on the global network.

From the perspective of a mobile user, Hyrax is easily set up as it mainly involves installing an Android application on the device. The only main setting that concerns users

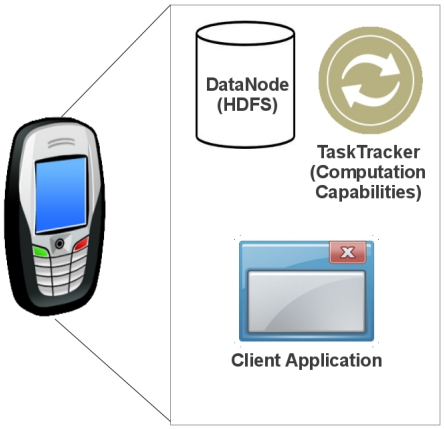

Figure 4.2: Mobile node within Hyrax

could be the IP address fo the master node to connect to. Other possible settings (for example, which files to share) would depend on the specific application that Hyrax is being used for.

From the perspective of the system, setting up Hyrax mainly involves installing the Hadoop code on a master node machine and running the NameNode and JobTracker daemons. The setting up of a cluster is therefore probably no more complicated than that of setting up a normal Hadoop cluster.

4.2 Porting Hadoop

We ported Hadoop 0.21.0 over to the Android platform for this implementation of Hyrax. The goal of porting the code was to create an Android application that could act as a slave node on a Hadoop cluster. In order to achieve this goal, the code for the DataNode and TaskTracker had to be completely ported over to the Android platform. This was a completely new port from that of the previous implementation of Hyrax, which ported Hadoop 0.19.1. The main challenge lay in reconciling the differences present in the Hadoop codebase that resulted due to the incompatibilities of the Android API and the Java API. We ported the code to run on devices that run Android 2.1 (Update 1) or later (ie Android API Level 7).

It is not our goal to use the Android devices as the master node in the Hadoop cluster. We therefore did not port the code for the NameNode and JobTracker over to the Android

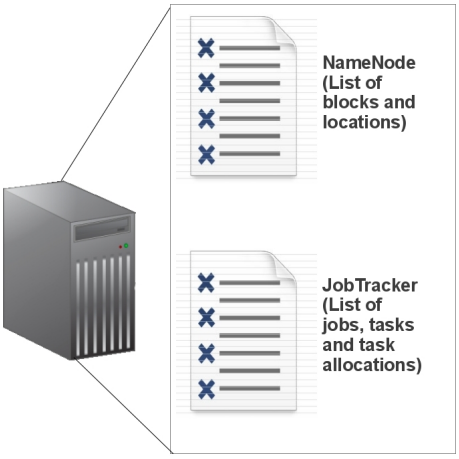

Figure 4.3: Master node within Hyrax

platform, although it should be possible to do so. We feel, however, that given the unreliability of the connection of a mobile node to the cluster, it is unsuitable to implement the NameNode and JobTracker on a mobile node as they are both single points of failure for the system. We thus still require a traditional desktop machine or server within a Hyrax cluster to serve as the master node.

4.2.1 Android Obstacles

As mentioned earlier, porting Hadoop over to the Android platform proved to be a difficult task. While API differences were easily caught and corrected at compile time, the sheer volume of code in the Hadoop codebase made this task of rewriting the offending portions of the code extremely time-consuming. Furthermore, besides the Hadoop codebase itself, some libraries that Hadoop made extensive use of (including Java libraries) also contained incompatibilities with the Android API. The source code for these libraries had to be downloaded, modified and included in the final Hyrax code.

Another major obstacle was that the `java` executable that is usually used to run Java programs is not available on Android. Instead, applications must be packaged in a `.apk` file before they can be executed in Android. This forces the entire Hyrax application to be packaged in the same file and to run in the same process. Since the original Hadoop launches the DataNode and TaskTracker instances as separate processes, this had to be changed for Hyrax. In Hyrax, the DataNode and TaskTracker instances are run as seperate

services within the same application.

A related issue is that of launching tasks on the Android slave nodes. Upon receiving a task from the JobTracker, Hadoop usually forks a new process and runs the task within that process. Forking a new process is, however, not possible with Android. The code for the TaskTracker has therefore been modified to launch a new thread to handle any new tasks that is assigned to it by the JobTracker. The new task is launched with an explicit call to the child's class `main` method.

A major improvement over the previous version of Hyrax is the ability of the current implementation to support dynamic class loading. Hadoop packages a job's code into a Java class file before sending it over the network (through the HDFS) to the slave nodes, which then load the classes dynamically and execute them at runtime. However, the byte-code used by the Java JRE and Dalvik are different. Since code must be in the `.dex` format before it can be excuted by the Android system, we are not able to dynamically load classes as Hadoop did without modifying the client side code that is used to launch the job. The client-side code has been modified to compile the job's code into the `.dex` format before it is sent to the slave nodes for execution. Of course, if job class files are to be launched from the mobile devices itself, it will probably already be packaged into the `.apk` file and not require any dynamic class loading, which will result in faster job execution.

Other issues include race conditions involving file I/O that are a result of different behaviours between the Java and Android libraries, where files are closed before they are completely written. These issues were more difficult to discover and debug because they could not be reliably reproduced and only manifested themselves at runtime.

It should be noted that due to the size of the Hyrax application, much of the debugging was done through observation of the application's log through the `logcat` utility provided with the Android SDK. This was a time-consuming and painful process as this meant that debugging was mostly done manually. Due to the size of the codebase, it also takes several minutes to compile Hyrax everytime a modification was made to the code. All these contributed to the development time of the entire system.

4.2.2 Hadoop Obstacles

Since Hadoop was designed to be run on traditional hardware, some of the assumptions made by its designers do not hold for mobile devices. Such assumptions affect the performance of Hyrax.

Assumptions regarding the amount of memory available for use can be easily corrected

through the Hadoop parameters. For example, Hadoop typically allocates buffers of up to 100 MB in size, but the heap space of an Android application is limited to 16 MB. The buffer sizes had to be reduced, which in turned cause excessive swapping to occur. This had to be reduced through the `io.sort.record.percent` parameter.

Another key change that was made was to change the format of the Hadoop configuration files from XML to a simple property file format. The primary motivation for this change was the excessive computation and memory requirements for parsing a XML file. This change would enable faster initialisation of the DataNode and TaskTracker instances, as well as jobs and tasks.

A few other minor changes were also made to the Hadoop codebase. An example includes a simple check when accessing data through the HDFS to see if the node where the data is to be retrieved from is the local node. This idea originated from [12]. The original Hadoop code treats all nodes the same, and forces the data to be retrieved through the Java networking APIs even if the data is available locally. The code was changed such that the data is retrieved from the node's local storage directly if the data is available locally. This simple change has led to a shorter execution time of up to 30 seconds on some sample example jobs provided with Hadoop.

4.3 Block Replication Strategy

The replication factor of each block of data in the HDFS can be configured. The replication factor is basically a tradeoff between the availability of data and the cost of network data transfer (and consequently, battery consumption). Having a high replication factor means that the data will be available on more nodes, which provides more flexibility in terms of task assignment as well as more tolerance towards node failure. However, it also means that each block of data will have to be copied to more nodes, resulting in higher network data transfer.

On the other hand, if the replication factor is low, then data is at risk of being lost from the cluster completely if the nodes that hold the relevant blocks suddenly depart from the cluster. Furthermore, if the data is in high demand, this also places a very high load on the devices that hold the blocks as other nodes will often request for these blocks to be read.

In order to mitigate the issues outlined above, users might prefer to specify a replication factor for each file (or type of file). For example, users might want to keep the replication factor for logs and multimedia files to be 1 so as to avoid excessive network transfers due to replications. In such a case, the file will only be stored on the local device where the

data originated from. However, if a file proves to be popular, it will make sense to give it a higher replication factor so as to reduce the load on the originating device, as well as to avoid excessive network transfer as the JobTracker has more flexibility in assigning tasks that require that data as input to nodes which already contain the relevant blocks. It will be quite trivial to allow users to set the replication factors for individual files as the HDFS allows this, so it is just a matter of exposing that setting to the users. Users might want control over this parameter on a case-by-case basis, depending on the requirements of their application.

Getting the block replication correct can be critical. On the one hand, we would prefer the replication factor to be low in order to minimise usage of disk space, since users are not likely to be willing to sacrifice too much disk space on their devices for Hyrax. A low replication factor can also result in the further benefit of less data transfers, since data needs to be replicated to fewer nodes. On the other hand, in order to ensure data availability in the face of node departures / failures, we prefer the replication factor to be high. Assuming that the probability of a node departing the cluster is d for every unit of time, and that the replication factor is r, then there is a probability of d^r that a block of data will be lost from the cluster if all the nodes that it is replicated to depart at approximately the same time (before the cluster has a chance of making up for lost replications). It is up to users or an administrator to determine what the acceptable tolerance for data loss is, together with a consideration of the amount of space needed to store data replications, in order to decide what the replication factor should be.

4.4 Network Organisation

Hadoop originally has a concept of servers on racks. Hadoop has a default replication factor of 3. In order to protect against data loss, Hadoop usually stores two copies of each data block on servers within the same rack, and the final copy on a server on a different rack. In this way, even if one entire rack of servers were to be unavailable (due to a power loss, for example), there would still be one copy of the data block on another rack, which will then be replicated to bring the number of copies of that block back to 3.

We do not currently have a concept of racks for Hyrax, although one could possibly be implemented. For example, we could possibly augment the mobile nodes with traditional servers so that data blocks could also be stored on those static servers, where the chances of node failure are far lower than that of mobile nodes. Furthermore, these servers will be able to process tasks in a shorter amount of time than a mobile node. In this way, we could possibly have two racks: a mobile rack and a server rack. This configuration

is feasible since very little code needs to be modified to adapt the Hyrax code to run on servers. This is because the code uses the same interfaces and network protocols for inter-node communications. It should be noted, however, that if static servers are present in the clusters, it would not make much sense to replicate data blocks to the mobile nodes anymore.

Another possibility would be to assign the mobile devices to racks, perhaps depending on their location in the network. However, due to the mobility of these devices, such a scheme could result in issues of frequently reassigning the devices to different racks as their locations change. This would in turn wreck havoc in the data replication scheme. Another possibility would be to assign racks to devices based on the kind of hardware and resources available on the devices. While this would ensure that once a device is assigned to a rack it will remain in that rack, it would result in uneven rack sizes of many of the devices are made up of similar hardware (same device model, for example).

In either case, a rack scheme, whether it be a combination of servers and mobile devices or just mobile devices alone, can be explored in future work.

Chapter 5

Evaluation

We present our evaluation results of Hyrax in this chapter. We mainly evaluate Hyrax based on its performance in relation to the requirements that were established earlier. Note that because we are using similar benchmarks as that of the previous study of Hyrax, we do not perform any analysis of the performance of Hadoop on traditional servers, as that analysis has already been done in the previous work.

5.1 Infrastructure

5.1.1 Testbed

We conducted our experiments on a pure mobile devices cluster (with the exception of the master node, of course) that consists of Samsung Nexus S [27] phones (see Figure 5.1). The Nexus S devices were all running Android 4.1.1, which was released on 9 July 2012.

The hardware of the Nexux S phones contain a 1 GHz Cortex-A8 processor, 512 MB of RAM and a 1500 mAh lithium-ion battery. The phone is capable of communicating through IEEE 802.11 b/g/n, has GPS capabilities, as well as sensors such as an accelerometer and a digital compass. In a departure from many earlier devices running Android, the Nexus S has an internal memory of 16 GB, with no allowance for the insertion of an external microSD card. This memory (known as external storage memory) is used to store not just user data (such as photo or video files), but it is also used to store the Hyrax logs, sensor logs, resource usage logs etc. Most importantly, it is also where the HDFS data blocks are stored.

Figure 5.1: Some of the Samsung Nexus S phones that were used as our testbed for Hyrax

The phones communicate with each other, as well as the master node, through a Linksys WRT54GL wireless router. The router is capable of supporting the IEEE 802.11 g/n standards. No modifications were made to the router.

The master node, which runs the NameNode and JobTracker daemons, runs on a desktop machine that is connected to the router through a normal Ethernet connection.

5.1.2 Benchmarks

We made use of benchmarks and run them on Hyrax in order to evaluate the performance of Hyrax. These benchmarks are derived from some of the example applications that come together with Hadoop. They include Sort, RandomWriter, PiEstimator, WordCount and Grep. We use similar benchmarks to that from the original Hyrax implementation study in order to provide a basis for comparison in any differences in performance between this new implementation over the previous one. The benchmarks are listed in Table 5.1.

As Table 5.1 suggests, the total input size for each benchmark is scaled to the size

Benchmark	Input Type(s)	Input Size
PiEstimator	Maps per host	3
RandomWriter	Bytes per map, Maps per node	1 MB, 2
Sort	Bytes per map, Maps per node	256 KB, 1
Grep	File size, Files per node	64 KB, 1
WordCount	File size, Files per node	32 KB, 1

Table 5.1: Benchmark input types and sizes per node.

of the cluster in order to ensure that each node in our cluster is assigned some work. Therefore, the total amount of work done by the cluster in aggregate will increase with the cluster size. However, the average amount of work done by each individual node within the cluster should be the same. We provide a brief description of each benchmark below.

In order to account for the base resource usage of Android's background processes, we also run a control benchmark by running Hyrax for 60 seconds without executing any jobs and collecting the corresponding resource usage statistics. This control benchmark will also account for the overhead of the DataNode, TaskTracker and communications with the master node.

We only take into account the resource usage statistics within the execution phase of each benchmark.

Description of Benchmarks

- The PiEstimator benchmark uses a Monte Carlo method to calculate an estimate for the value of π.
- The RandomWriter benchmark randomly generates a certain amount of data at each node and writes it to the HDFS.
- The Sort benchmark actually makes use of the RandomWriter benchmark to first generate sortable data, and then sorts this randomly generated data. Sort's map phase is an identify function, as the benchmark takes advantage of the fact that intermediate keys are sorted by Hadoop before the Reduce phase.
- The Grep benchmark searches for a word or pattern within a given piece of text placed on the HDFS.
- The WordCount benchmarks counts the number of times each individual word appears in a given piece of text placed on the HDFS.

We launch each benchmark from a separate client machine (not a mobile device) through the `./hadoop` program and leave Hyrax to load the required code to execute the jobs from the HDFS dynamically.

5.1.3 Analysis tools

We make use of both Android system resource usage logs and Hadoop logs in order to analyse the performance of Hyrax.

System resource usage logs

We make use of information from `/proc` to study the system resource usage of Hyrax. We study relevant information such as CPU usage, memory usage, disk I/O and network I/O. To aid us in this, we made use of code from NetMeter, an Android application that allows users to troubleshoot performance issues by collecting network and CPU usage statistics over time.

We also included code to log the battery levels of each device over time in order to study the battery consumption due to Hyrax. Naturally, we have to make sure that the devices are unplugged from any chargers or USB ports in order for the battery level statistics to be meaningful.

5.2 Baseline performance of mobile devices vs. traditional servers

A study of the differences in the baseline performance between an Android G1 and traditional servers was already done in the previous report on Hyrax. We now explore the baseline performance difference between the Samsung Galaxy S, Android G1 and traditional servers by running the same four micro-benchmarks on the Galaxy S and comparing the results with that from the previous work.

The four micro-benchmarks are each bound by CPU, memory, disk or network resources. A brief description of each benchmark is given below.

Benchmark	G1 (MB/S)	Nexus S (MB/s)	Server (MB/s)	Nexus S Advantage
Memory Write	12	142	4600	11.8x
Memory Read	11	114	4500	10.4x
Disk Write	8.7	5.5	66	0.6x
Disk Read	15	16	460	1.1x
Network Write	0.92	0.26	87	0.3x
Network Read	0.64	0.2	86	0.3x
CPU	N/A	N/A	N/A	6.6x

Table 5.2: Results from Simple Benchmarks on Nexus S

Descriptions of Micro-Benchmarks

- The CPU benchmark simply executes an empty loop for a number of iterations.
- The memory benchmark writes to a buffer sequentially for a number of times, and then it is subsequently sequentially read from.
- The disk benchmark, data is sequentially written to the device's internal memory andsubsequently read.
- The network benchmark runs the benchmark on one device while a socket is running on another similar device. Data is written and read from the socket server.

The results of the simple benchmarks on the Nexus S are given in Table 5.2. We also include the results from the G1 and traditional servers from the previous report for comparison. In this table, the Nexus S advantage is the advantage of the Nexus S over the G1.

As Table 5.2 shows, the newer hardware and software of the Nexus S offers advantages over the Android G1 in terms of memory access and CPU speed. The performance of disk access is comparable between the two devices. However, the Nexus S performs poorly in terms of network I/O. This has implications for Hyrax since Hyrax depends on data and code being transferred over the network. Furthermore, the poor performance of network I/O in both devices suggests that it would be prudent to keep the need of network I/O in Hyrax down to a minimum.

The huge difference in memory access is probably due to a combination of better hardware as well as better software in the Nexus S. The better performance of the CPU is not surprising since the Nexus S has a more capable processor than the Android G1.

5.3 Performance Improvements in Hyrax

In this section, we study the performance differences between this implementation of Hyrax agains the previous one. We also determine how Hyrax scales with the cluster size.

5.3.1 Question

We aim to explore any improvements (or setbacks) from the previous version of Hyrax.

Approach

We run the exact same benchmarks on Hyrax as that in the previous study in order to provide a basis for comparison. We varied the cluster size from 1 to 9 and ran each experiment 5 times and took the average values of statistics such as job execution time, task execution time, CPU utilisation etc. Similar to the previous study, we set the replication factor of the HDFS to 2 in each case.

Recall from Subsection 5.1.2 that in our benchmarks, the total amount of work done by the cluster in aggregate increases with the size of the cluster, but the average amount of work done by each individual node remains approximately the same.

Since the benchmarks used are the same, a direct comparison can be made between the results of this study and those of the previous one.

5.3.2 Hypothesis

Our hypothesis is the same as that in the previous study.

According to Amdahl's Law, we can expect the total execution time of a benchmark to increase linearly with the cluster size, depending on what fraction of the benchmark can be executed in parallel.

Amdahl's law states that given n nodes, we can expect a maximum speed-up of

$$S_n = \frac{1}{(1-P) + \frac{P}{n}}$$

, where P is the fraction of a program that can be parallelised.

If we apply Amdahl's law and assume a fixed input size, then we can derive a lower bound on our execution time. This should be

$$E_n \geq ((1 - P) + \frac{P}{n})E_1$$

, where E_n is the execution time for a cluster of size n. However, since the input size for our benchmarks are actually proportional to n, we have to modify the above inequality. Given that the input size, I_n of our input size is

$$I_n = nI_1$$

, we can therefore determine the execution time per unit of input instead. This will be

$$\frac{E_n}{I_n} \geq ((1 - P) + \frac{P}{n})\frac{E_1}{I_1}$$

We can then derive that

$$E_n \geq (1 - P)E_1 n + PE_1$$

In other words, we can expect that the execution time for our benchmarks will increase linearly with a gradient of $(1 - P)E_1$. This depends on the value of P, which is difficult to determine. Figure 5.2 shows the predicted execution time versus cluster size for various values of P.

We expect that average CPU utilisation should remain consistent across cluster size, since the average amount of computation performed by each device is about the same regardless of cluster size. We, however, expect that network IO will increase roughly linearly with the cluster size since our input data size is proportional to the cluster size.

5.3.3 Results

Execution time

The exeuction time of a benchmark is the total time that a job takes to run in a benchmark. These job times are calculated from the job launch time and the job finish times in the Hadoop logs.

In general, the execution time of a job increases with the number of clusters n. This can be seen in Figure 5.3 and Figure 5.4. As predicted by the model, the execution times of the tasks increase roughly linearly with the cluster size. However, we note that the absolute execution times of the Sort benchmark is higher than that from the previous study, while those of the RandomWrtier benchmark are similar.

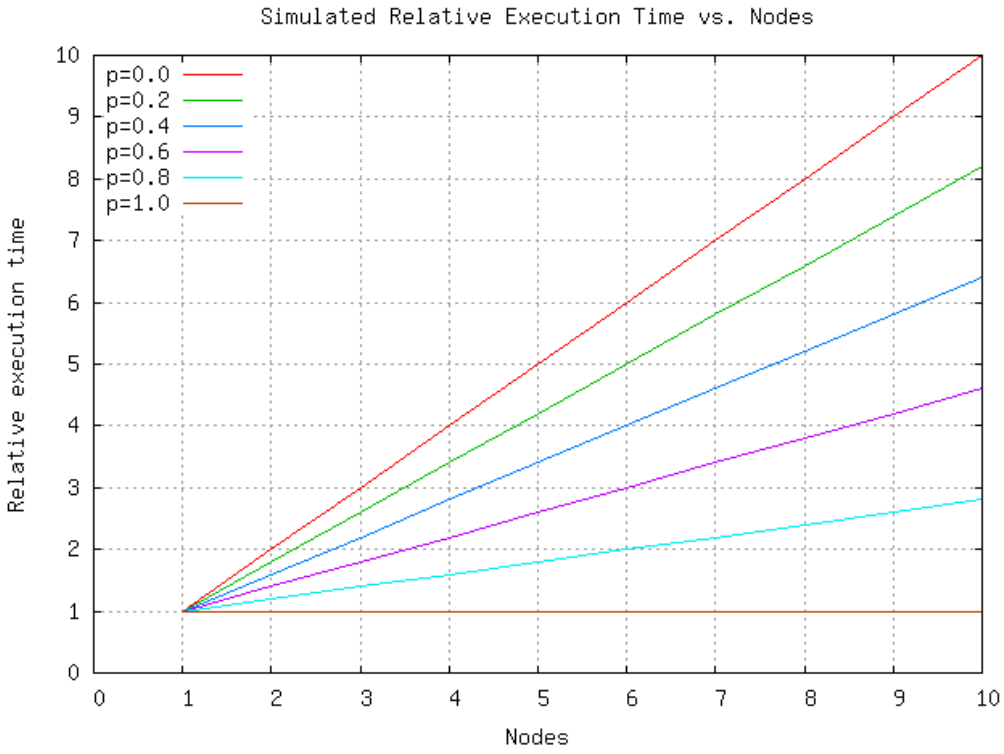

Figure 5.2: Simulated relative benchmark execution time vs. number of nodes for varying levels of parallelisation.

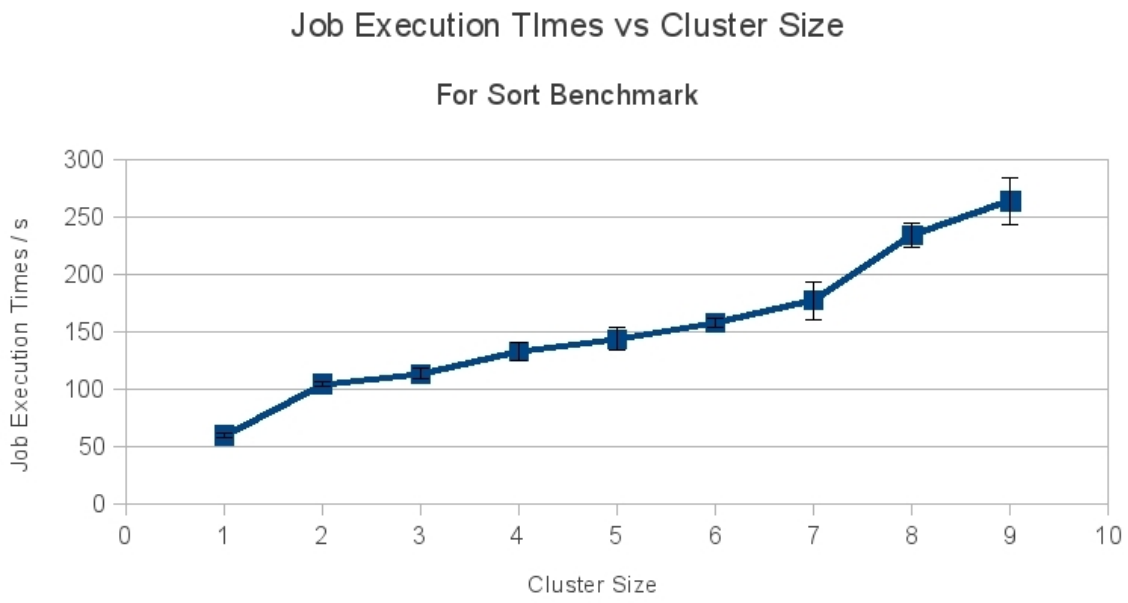

Figure 5.3: Average execution times for the Sort benchmark on different cluster sizes

Resource usage

Figure 5.5 shows the average CPU utilisation for the control benchmark, while Figure 5.6 shows the average network I/O for the control benchmark. Recall that the control benchmark is simply a benchmark that runs Hyrax for 60 seconds without running any jobs in order to determine how much the CPU is utilised by Hyrax and other applications on the device.

As can be seen in Figure 5.5, the average CPU utilisation stays relatively constant for all cluster size, which is to be expected since no jobs were executed, therefore there was no unusually heavy demand for the CPU. The same can also be said for the average network I/O for the control benchmark, where the average number of kilobytes sent and received per device stays relatively constant for all cluster sizes. This means that the average background network traffic required to run Hyrax does not depend on the cluster size.

We expect CPU utilisation to increase when running our benchmarks, but it should remainly relatively constant across cluster sizes. For network IO, we expect network IO to increase roughly linearly with the cluster size since our benchmark input size is proportional to the cluster size. Furthermore, since we are running Hyrax on a closed network,

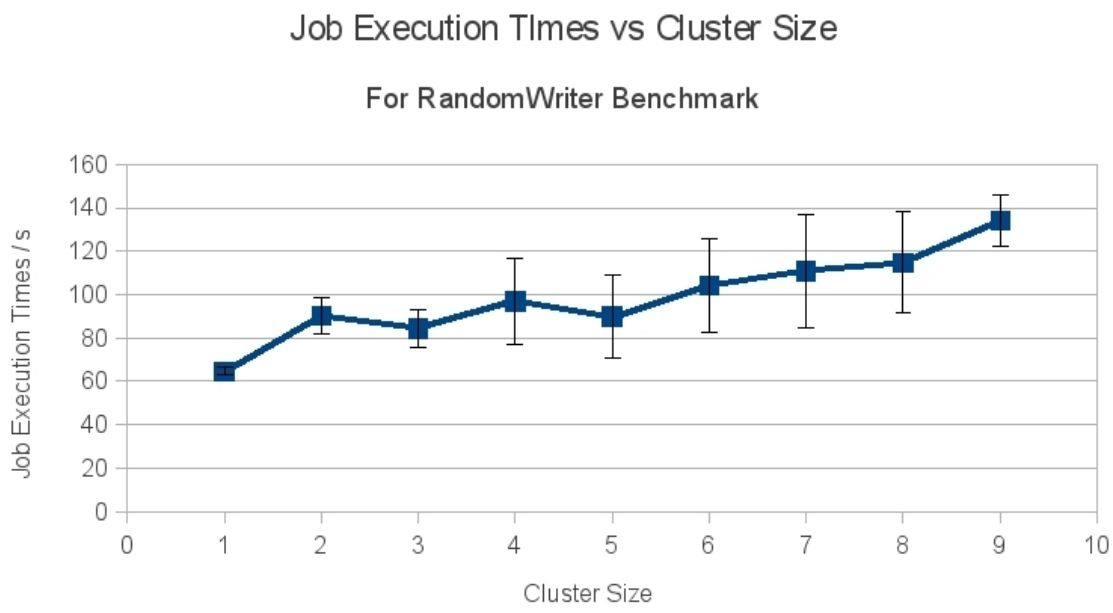

Figure 5.4: Average execution times for the RandomWriter benchmark on different cluster sizes

the total bytes received over the network should roughly correspond to the total bytes sent.

We also show the average CPU utilisation for the Sort and PiEstimator benchmarks in Figure 5.7 and Figure 5.8 respectively. Compared to the previous study, the average CPU utlisation for both benchmarks is lower in this study, most probably due to the better hardware that we used since the benchmarks and input sizes were the same.

Surprisingly, if we compare Figures 5.7 and 5.8 with Figure 5.5, it appears that even with jobs executing, the load on the CPU does not increase substantially at all. Of course, CPU load is also job-dependent, but a further study of the CPU load imposed by jobs may be warranted.

Figure 5.9 shows the total network IO for 5 runs of the Sort benchmark. As expected, the total network IO increases with the number of nodes due to the increase in the amount of data to be sorted and also increased background traffic. However, given that the total amount of data that needs to be sorted is only $2n$ MB, where n is the number of nodes in the cluster, the figures in Figure 5.9 seem unusually high. For example, in the case of 2 nodes, an average of over 12 MB of data is transmitted over the network just to sort 4 MB of data for each run. The situation worsens in the case of a cluster consisting of 9 nodes, where sorting 18 MB of data results in traffic in excess of 100 MB per run. Even if the

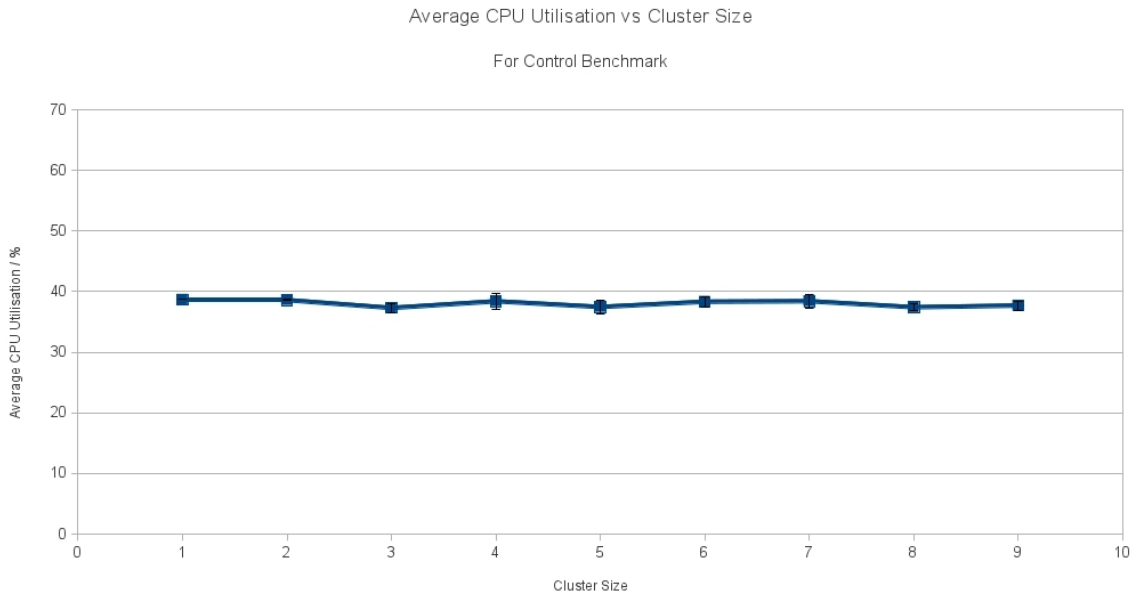

Figure 5.5: Average CPU utilisation for the control benchmark

figures in Figure 5.6 are taken into account, the high network IO for the Sort benchmark is still not sufficiently explained. A further investigation into whether this constitutes an inefficiency in either the HDFS or MapReduce framework is warranted in a future work.

5.3.4 Conclusions

As expected, the average job execution time increases with the cluster size. This is in line with our hypothesis involving Amdahl's Law. The average CPU utilisation per device also remained fairly consistent, which is also in line with our hypothesis. The trend of total network IO also agreed with our hypothesis, with total network IO increasing roughly linearly with the cluster size. However, the absolute figure of the total network IO does not seem to correspond with the input data size. The total network IO figures in Figure 5.9 seem excessive when compared with the input size of the job, which is approximately $2n$, where n is the cluster size.

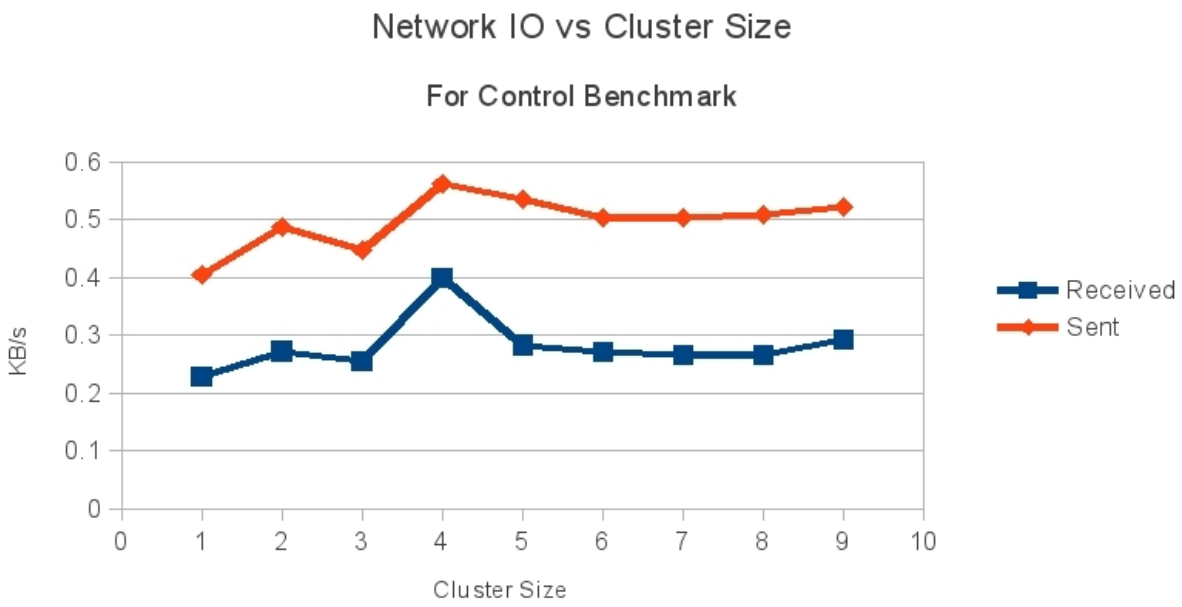

Figure 5.6: Average network I/O for the control benchmark

5.4 File sharing

One of the primary motivations for developing Hyrax was to avoid uploading data to offsite cloud services over the Internet, especially when the data is readily available from devices that are in the vicinity. We now evaluate how effective Hyrax and the HDFS is in achieving this aim by comparing the file upload and download times to and from both the HDFS and a file hosting service over the Internet.

5.4.1 Question

Our main goal is to compare how file sharing (specifically, file upload and download) within the HDFS compares with file sharing over a file hosting service through the Internet. Our secondary goal is to compare how the replication factor of files within the HDFS affects the file upload and download times.

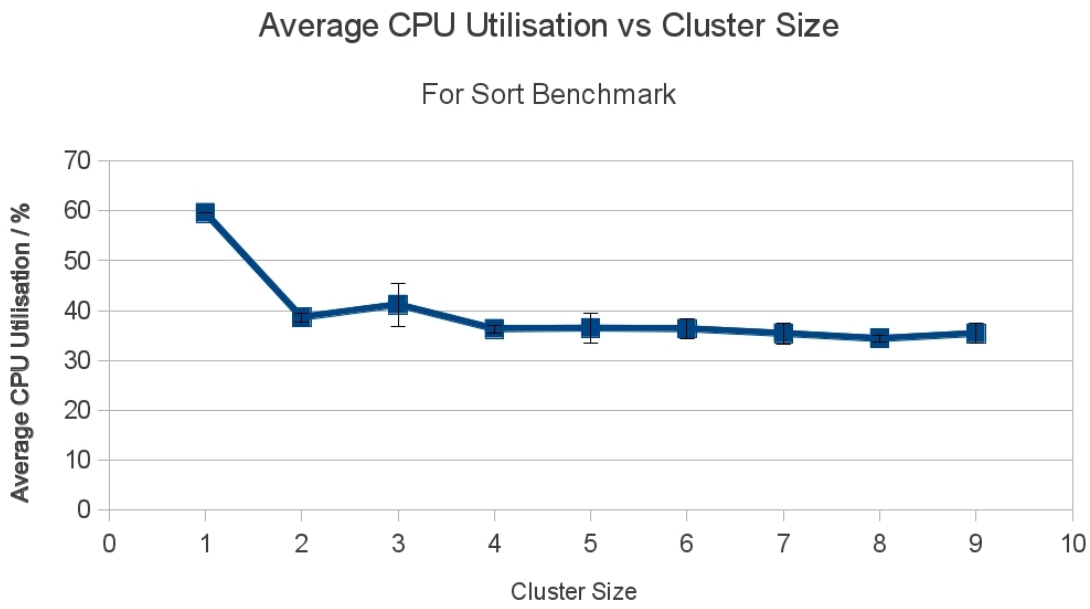

Figure 5.7: Average CPU Utilisation for Sort benchmark

5.4.2 Approach

We make use of DropBox [21], a popular file hosting service online, as our offsite file hosting service. We access the Internet through a campus wireless connection. The files were uploaded and downloaded through the DropBox application for Android. We chose to use an existing online file hosting service instead of our own server as transferring data over the Internet appears to be a more realistic scenerio since most mobile users currently access cloud services over the Internet.

To determine file upload times, we upload files of sizes $5, 10, 15, 20, 25$ and 30 MB to the HDFS and the offsite file hosting service from a node within a cluster of 5 nodes. For the HDFS, we do this for every node and take the average of the readings. We also vary the replication factor of the files. For uploading to DropBox, we only take the reading of one upload as DropBox appears to keep a copy of the file on their server even after we have deleted it, which makes for inaccurate readings for subsequent uploads.

We keep the cluster size constant at 5 when uploading files. We do not anticipate that there will be significant changes in file upload times regardless of cluster size. We conduct an evaluation of an application that studies file upload times with respect to cluster size in Subsection 6.4.2. The results from Figure 6.5 seem to suggest that it is indeed the case

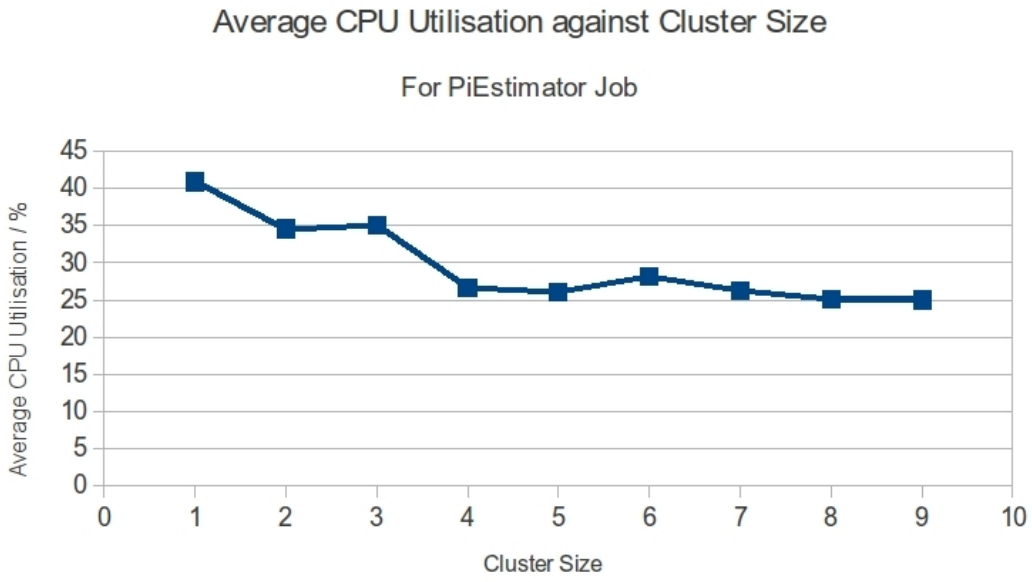

Figure 5.8: Average CPU utilisation for PiEstimator benchmark

that cluster size has no significant impact on file sharing operations.

File download times are measured by downloading the same file from all nodes within the cluster simultaneously. We vary the cluster size between 3 to 7 as well as the replication factor of the files to determine how fast the HDFS can serve data to multiple nodes simultaneously.

Data Set

Our files are generated through the `dd` utility, with input taken from `/dev/random`. We use this to generate files of sizes $5, 10, 15, 20, 25$ and 30 MB.

5.4.3 Hypothesis

For uploads to the HDFS, we expect the upload times to increase with the replication factor. This is due to the additional copying that needs to be performed when data is written to the HDFS. On the other hand, we expect the download times to decrease as the replication factor is set higher. This is because with a higher replication factor, less data

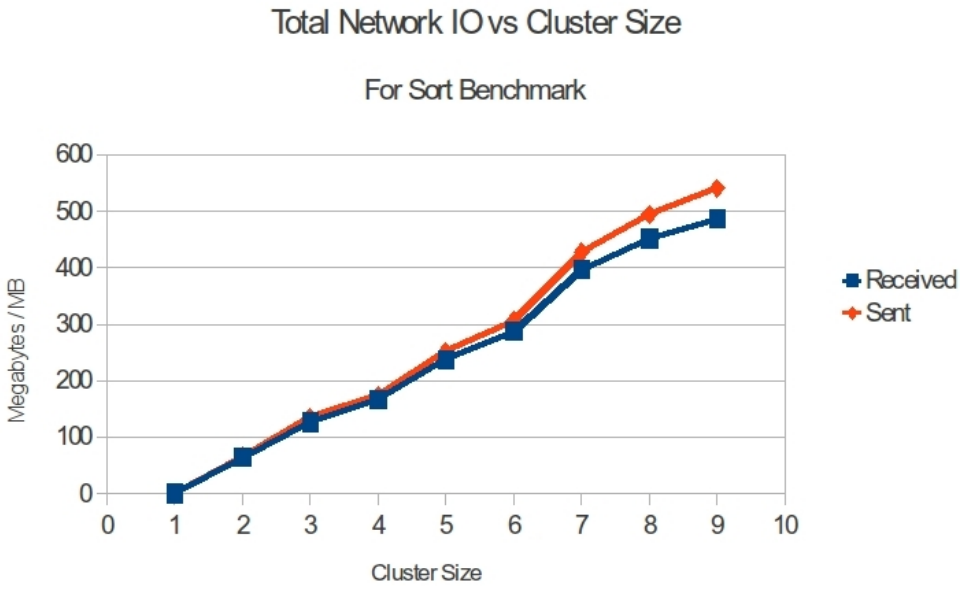

Figure 5.9: Total Network IO for 5 runs of Sort benchmark

transfer is required overall due to more nodes already having the data locally.

We also expect that the transfer times will increase with file size for both file uploads and downloads since more data needs to be transferred when files are larger in size.

It is hard to predict how the transfer times to an online hosting service will compare with that of the HDFS. We expect that the HDFS should outperform the online service for a replication factor of 1. However, due to the additional data transfer required for higher replication factors, it is possible for the online hosting service to outperform HDFS in those cases.

We expect the total amount of network traffic to increase with the file size as well. The increase should correspond to the number of times the file is replicated on the network. For example, for a replication factor of 2, we expect that the total network IO should be around the same as each file size, since the file will only be sent over the network once.

5.4.4 Results

File Upload

The results for our file upload experiments for files of different sizes and different replication factors are shown in Figure 5.10.

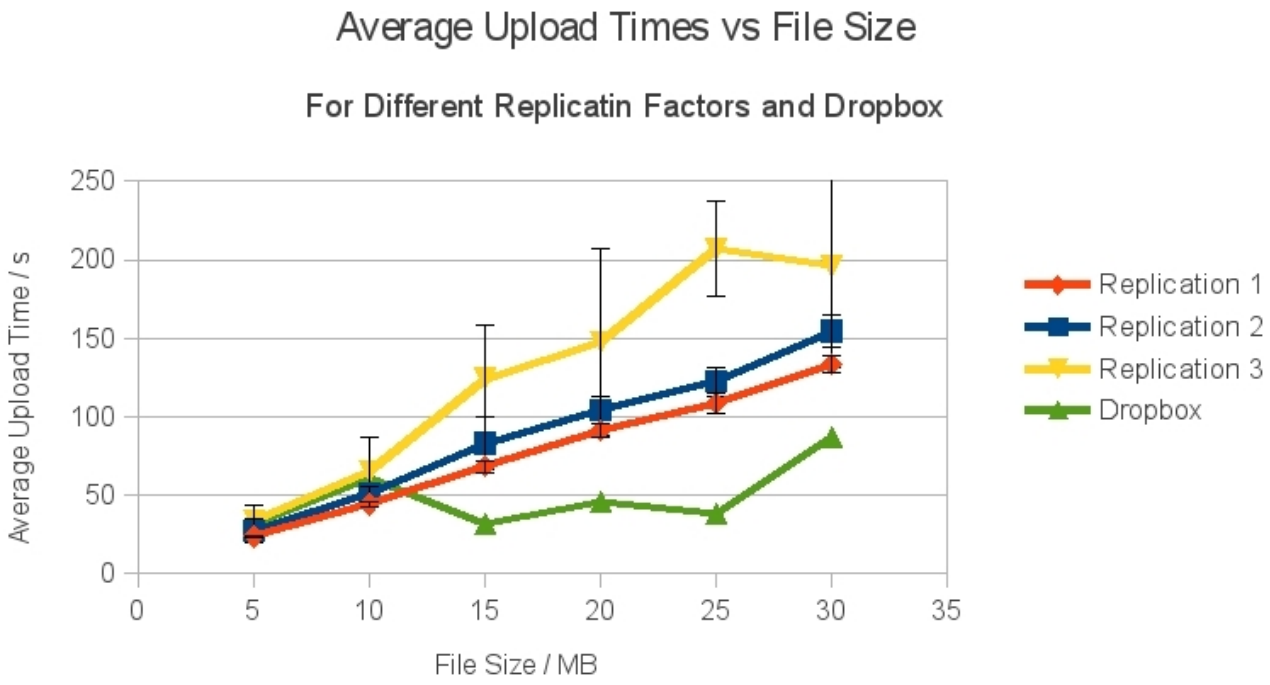

Figure 5.10: Graph of Average File Upload Time against File Sizes for a Cluster Size of 5 with Standard Deviation

Figure 5.10 suggests that our hypothesis that file upload times will increase with replication factor is correct. It also validates our hypothesis that larger files will result in longer file upload times. However, the standard deviation, especially that for a replication factor of 3, suggests that file upload times can vary widely across different runs and devices. This could be due to unreliable network links, resulting in more time spent repeating transmissions.

Figure 5.10 also shows that uploading files to DropBox over the Internet will take a shorter amount of time, especially for larger file sizes. Since our network latency is shorter when transferring to the HDFS than by transferring across the Internet, this result possibly exposes some of the weaknesses or inefficiencies of the HDFS or its data transfer protocol.

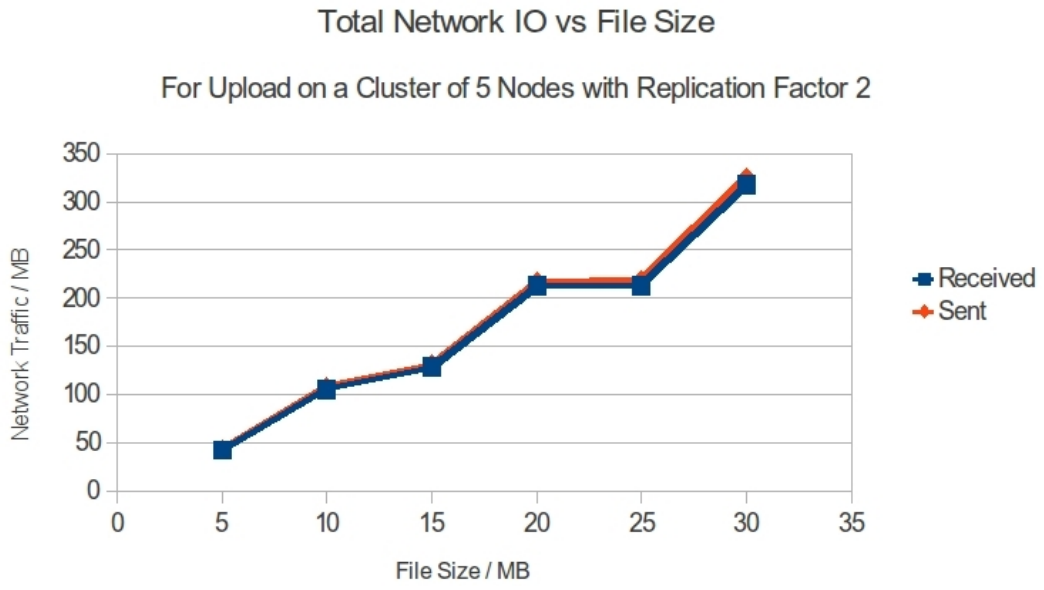

Figure 5.11: Graph of total network IO for a cluster of 5 nodes and replication factor 2

Figure 5.11 shows the total network IO for uploading the files on a cluster of 5 nodes with the replication factor set to 2. This is for a total of 5 runs of the uploading experiment, so we should expect to see a total network traffic (that is, either received or sent) of about $5s$, where s is the size of the file being uploaded. Instead, we see that the total network traffic is way more than $5s$, sometimes up to 2 times. This does not fit with our hypothesis, but agrees with our findings from Subsection 5.3.3.

File Download

The results for our file download experiments for a 10MB file are shown in Figure 5.12.

The average download time of a 10MB file from DropBox for a cluster of 7 phones was about 20 seconds.

Figure 5.12 suggests that our hypothesis of longer download times for smaller replication factors is correct. Furthermore, the average download time increases with the cluster size, which is also expected since more bandwidth is bring used to transfer the blocks to the requesting nodes. The small standard deviations seen in the download times also suggests that download times are consistent across runs and nodes.

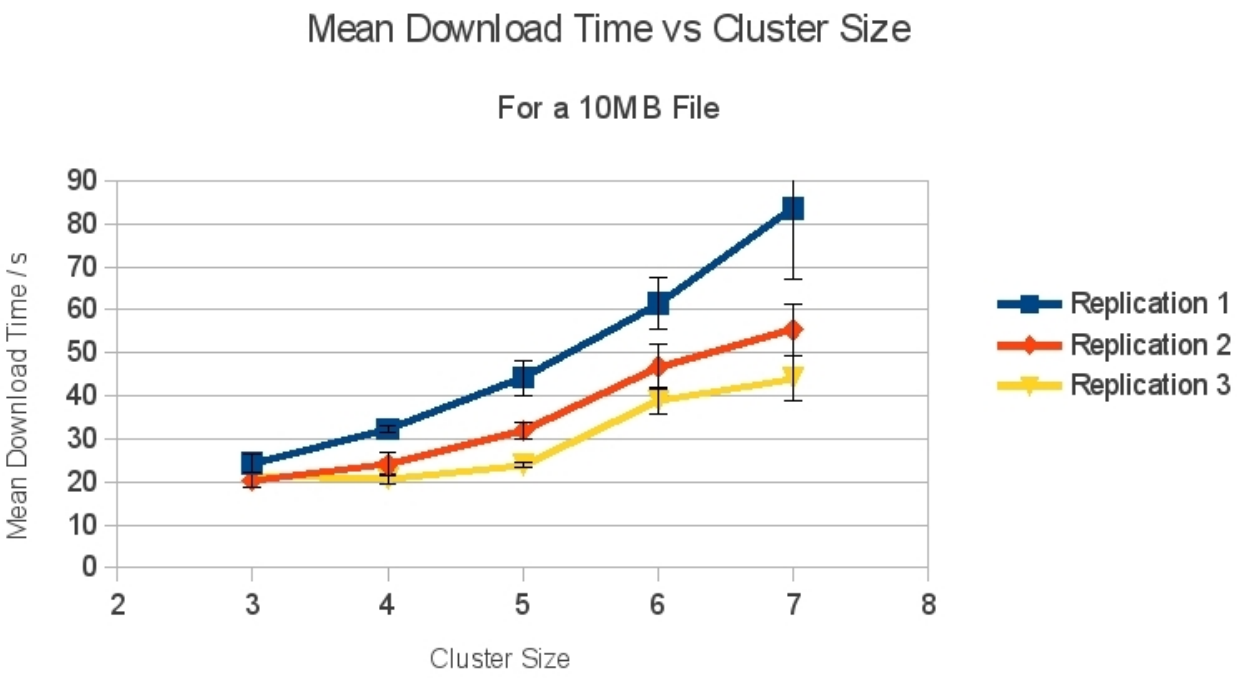

Figure 5.12: Graph of Average File Download Time against Cluster Size for a 10MB file with Standard Deviation

5.4.5 Conclusions

While setting a higher replication factor will result in increased parallelism in serving data, it comes at a cost of higher data transfers during data upload as compared with uploading to a server offsite. However, a faster download time might offset the high cost of file upload with a higher replication factor, depending on application. Also, note that the Internet connection used for this experiment is fast as it was a campus connection. Internet connections of similar quality might not be assured in a more realistic situation.

Again, as with Subsection 5.3.3, we see excessive network IO traffic when uploading the files to HDFS. This excess traffic could possibly account for the slow upload speed that was observed. A further investigation into the causes for the excess traffic could be undertaken in the future.

5.5 Battery consumption

Power resources are a huge constraint on any mobile device. Users will not want to use a system or application that consumes too much power, resulting in the need for more frequent charging. In this experiment, we investigate the rate of battery consumption on our testbed equipment while running Hyrax, and compare the results to that from the previous implementation.

5.5.1 Question

In this experiment, we want to know how does the battery consumption of the current implementation of Hyrax compare to that of the previous implementation.

5.5.2 Approach

Similar to the previous study, we run the Sort benchmark so as to provide a basis for comparison. The benchmark is ran repeatedly until battery exhaustion.

Besides collecting statistics on battery levels periodically, we also collect logs of other system resources such as CPU utilisation, memory use as well ask Disk I/O. We also collected the logs from the jobs in order to determine battery consumption rate during different phases of a job (map, reduce etc).

We run the experiments on clusters of 3 nodes, 5 nodes and 7 nodes.

Note that the GSM modules of the phones were left running, as would be typical of a phone usage scenerio, even though no calls nor SMS messages were sent / received while the experiments were running. Therefore, the GSM modules would have consumed some of the battery power. Note also that the screens of the phones were turned off for the duration of the experiments, which would also be typical of normal usage as we do not anticipate that users would have their screens on while running Hyrax unless they are performing other activities or are acting as the client.

5.5.3 Hypothesis

Carroll and Heiser [13] have shown that RAM and SDCard usage play a negligable role in the overall power consumption of a smartphone. On the other hand, they have concluded that the screen and GSM module are the biggest consumer of battery power. However,

since we run out experiments with the screen turned off and the GSM module in what is essentially an idle state, we believe they are not significant contributing factors to power usage for our experiments.

Given that we are using newer hardware (Nexus S versus Android G1) with a battery that can hold more charge (1500 mAh versus 1150 mAh), we would expect that the batteries should be able to last longer in this study than in the previous one. Furthermore, we also expect reduce tasks to consume more power than map tasks since reduce tasks involve network I/O during the "sort" and "shuffle" phases of the task. We expect power consumption to increase with the number of nodes in the cluster as well since an increase in the number of nodes means an increase in network traffic due to increased communications between nodes.

5.5.4 Results

Figures 5.13, 5.14 and 5.15 show the graphs of the battery levels over time for a cluster of 3 nodes, 5 nodes and 7 nodes running the Sort benchmark respectively.

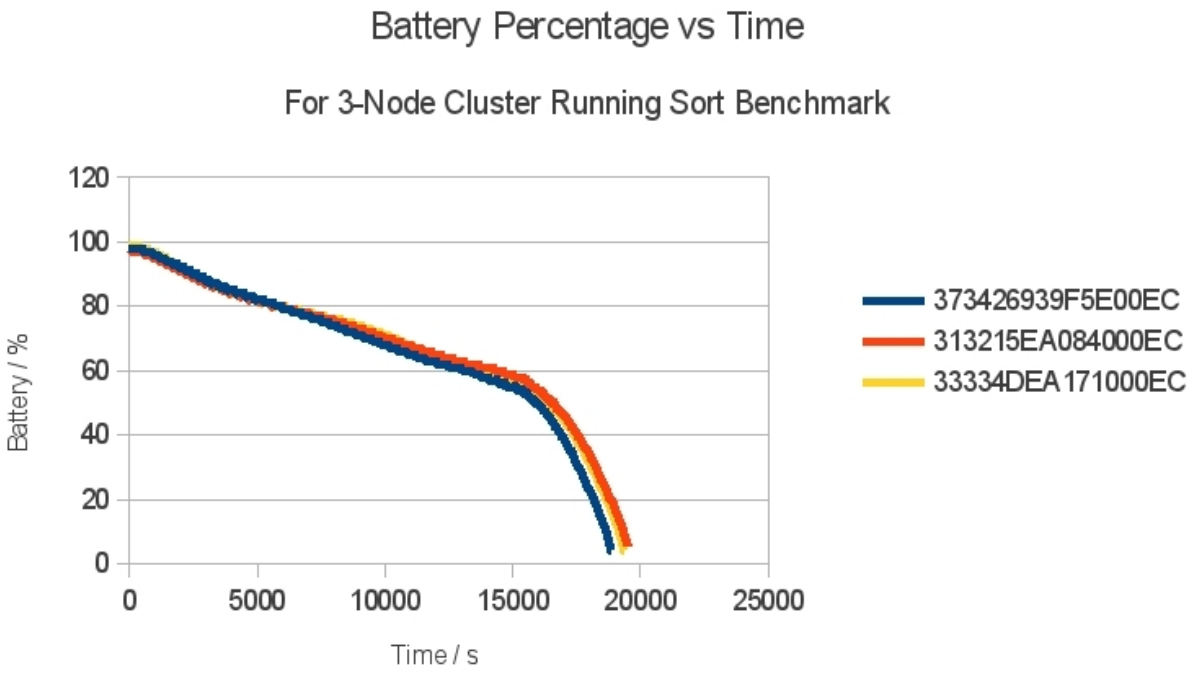

Figure 5.13: Battery consumption for the Sort benchmark on a cluster of 3 nodes

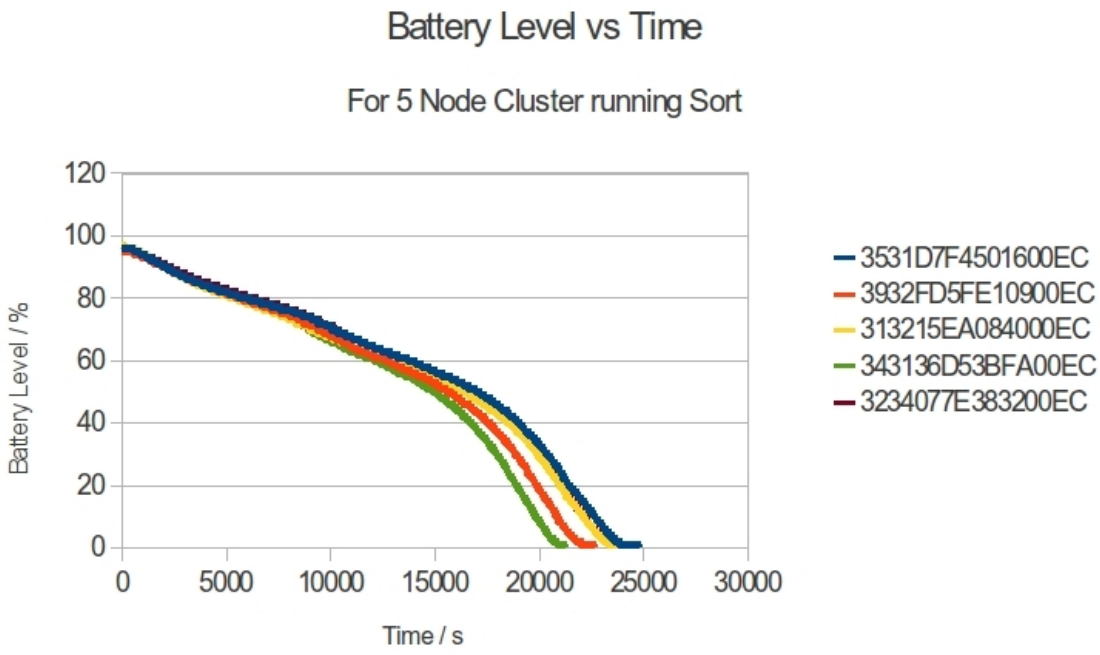

Figure 5.14: Battery consumption for the Sort benchmark on a cluster of 5 nodes

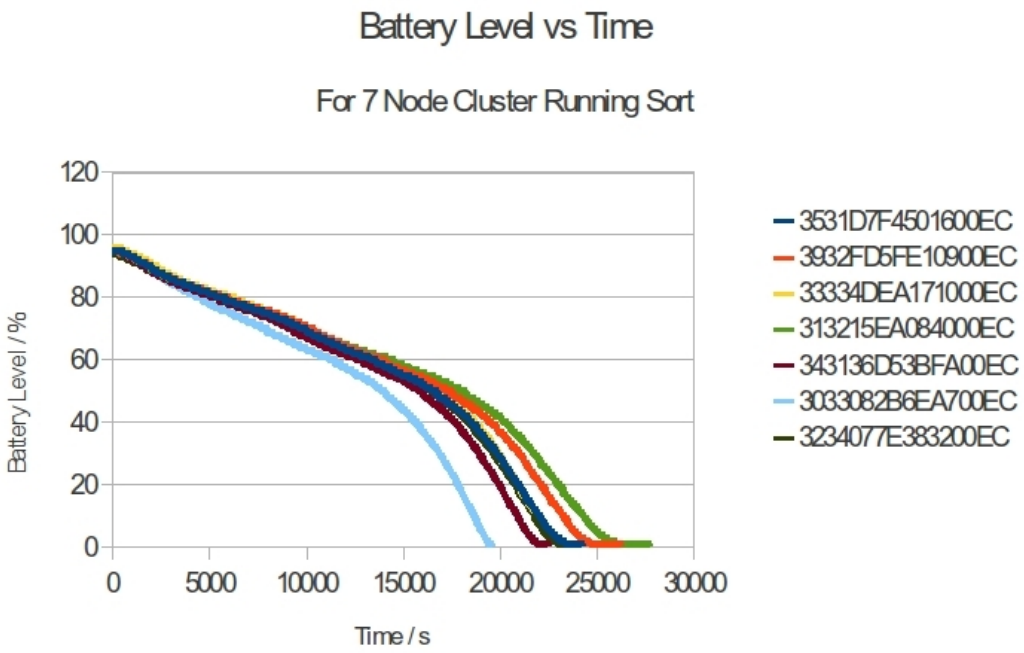

Figure 5.15: Battery consumption for the Sort benchmark on a cluster of 7 nodes

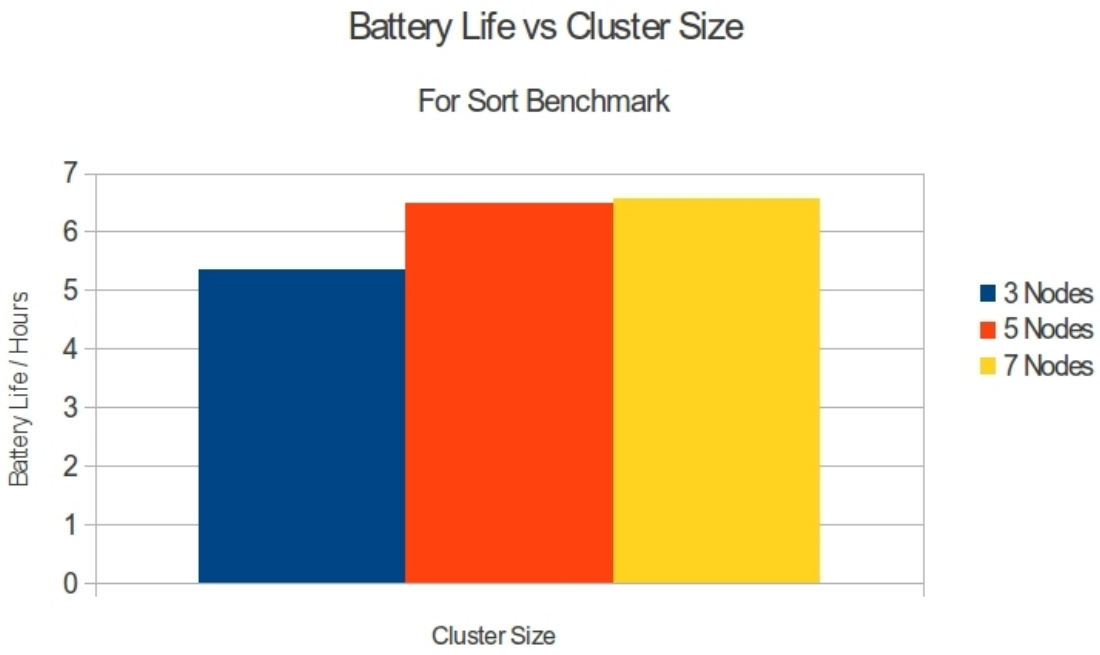

Figure 5.16: Average Battery Lives for Sort Benchmark

Resource	3 Nodes	5 Nodes	7 Nodes
CPU	23.58 %	37.60 %	34.15 %
Disk reads	0.160 reads/s	0.103 reads/s	0.109 reads/s
Disk writes	6.046 writes/s	5.761 writes/s	5.025 writes/s
Network send	36.00 KB/s	41.94 KB/s	37.89 KB/s
Network receive	34.55 KB/s	39.74 KB/s	35.21 KB/s

Table 5.3: Mean resource usage for each battery workload. Computed over entire duration of each workload and averaged over all phones.

Figure 5.16 shows the average battery life for each device for clusters of size 3, 5 and 7 running the Sort benchmark.

Table 5.3 shows the resource usage of each run on 3 nodes, 5 nodes and 7 nodes. These readings are the mean readings over all the phones for each cluster.

It should be noted that although the average network traffic appears low in Table 5.3 overall, a previous study by Balasubramanian et al [9] has shown that the energy consumption for WiFi is highly dependent on the inter-transfer interval. They show that WiFi is energy inefficient when used for small sized transfers. Hence, even if the average network traffic is low, if the inter-transfer interval is high, the overall energy consumption due to WiFi traffic might still be higher than a traffic load of higher transfer size but lower inter-transfer interval.

Furthermore, it appears that CPU utilisation is also highly dependent on hardware. For example, for the 5-node experiment, the average CPU utilisation of the Nexus S phones was only 37.6%, whereas the average CPU utlisation of the Android G1 phones was over 70%. With a lower average CPU utilsation, devices with better processors appear to be more capable of handling the requirements of Hyrax while still retaining the ability in terms of computation resources to handle other tasks as required by the user. Futhermore, depending on the individual processors, it is possible that a lower CPU utilisation will result in lower power consumption as well.

5.5.5 Conclusions

As expected, the average battery lives are slightly longer than those in the previous study. However, the average battery life of a device running the Sort benchmark appears to increase slightly with the cluster size, which is contrary to what we expected. This can be seen in Figure 5.16. The reason for the increase is not immediately clear, as the resource

usage (CPU utilisation, network IO and disk IO) don't appear to be significantly different across cluster sizes either, as shown in Table 5.3. We believe, however, that further increases in battery life can be achieved in the future. One area for possible improvement is the amount of data that is transmitted across the network, which currently appears excessive given the size of input data for the jobs.

Chapter 6

Case Study: A Distributed Music Search and Sharing Application

In order to showcase the potential of Hyrax, we developed a simple music search and sharing application on top of it and evaluated its performance. This application, known as *MusicDJ*, allows users to share the music files stored on their devices with others within their vicinity. Such an application could easily be extended to other types of files such as image and video files. It would enable users to obtain such files without having to perform a search on the Internet.

MusicDJ has two basic functions: To allow users to select a folder containing music files on their local device to upload to the cloud, and to allow users to search for a particular song or music file on the cloud by either title or artist, download and play it. The search functionality is provided by the MapReduce framework, which searches through an index of all the available files through a distributed job. We implemented the user interface of *MusicDJ* as an Android application.

6.1 Requirements

We established the following requirements for *MusicDJ*:

1. Users must be able to upload music files to the cloud and search for music files that are stored on the cloud. Users should be allowed to search for files by song title or artist.

2. Users must be able to download any music file that is stored on the cloud to their local mobile device.

Furthermore, on the backend, the search functionality of *MusicDJ* must be able to scale with the number of devices and files that are available on the cloud. It should also provide reliable storage for the music files and indices. These properties are already provided by Hyrax, as they were assumptions and requirements that were inherent in the design of Hadoop.

6.2 Design and Architecture

The design of *MusicDJ* was deleberately kept simple so that it would be easy to implement and evaluate. It is a proof-of-concept application that is not meant for public deployment as it depends on each device to supply and maintain the correct information about each music file that was uploaded. In other words, there is no mechanism to verify that the information that is stored on the index files are correct and up-to-date.

The high-level design of *MusicDJ* is shown in Figure 6.1, which is similar to Figure 4.1 in Chapter 4. More detailed illustrations of both the cloud component of the system as well as the individual devices specific to *MusicDJ* are shown in Figure 6.2 and Figure 6.3 respectively.

The key to the search and download functionalities of *MusicDJ* is the index files. These are simple text files that are generated by each device when users upload music files to the cloud. These text files contain information about the title, artist, duration and storage location within the HDFS for each music file that was uploaded by the user. Each line of the text files correspond to one music file. The index files from each device are then stored in a special directory within the HDFS (`/indices`). Music files that are uploaded to the HDFS by users are also stored in device-specific directories on the HDFS.

In order to execute a search, a MapReduce job is started with the user-supplied search term as input. The map phase of the job reads through every single line of all the index files in the `/indices` directory, and performs a regular expression match of the search term with the title and artist of each file specifiedin the index files. The map phase only emits an output if there is a match. The reduce phase of the job simply collects all the intermediate output from the map phase and outputs them into one final output file, which is then copied to the client device before being processed locally to display the results of the search to the user.

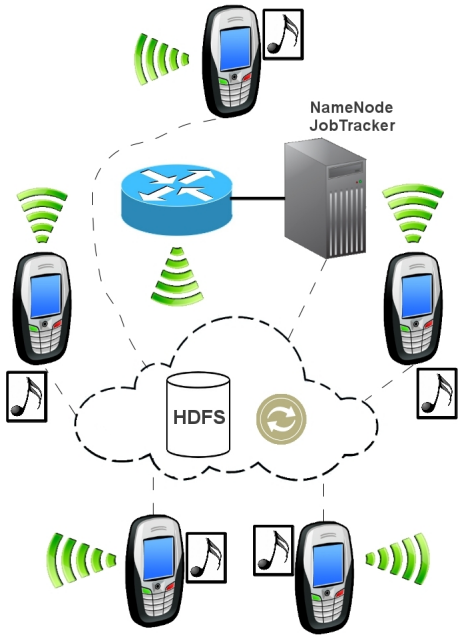

Figure 6.1: Overview of design of *MusicDJ*. Each mobile device contains its own music files on its own local filesystem

It is important to note that while logically the music files and index files stored on the HDFS are all "in the cloud", physically they all reside within the disk space of the individual mobile devices that make up the cluster. The "cloud" as shown in Figure 6.1 only exists due to the virtualisation of the individual mobile devices as a single computation and storage unit.

6.3 Implementation

The backend (upload, search, download) of *MusicDJ* did not take much effort to implement as most of the low-level details of storage location and task distribution were handled automatically by the Hadoop system. The main application code itself has absolutely no knowledge of the physical details of the cloud service that is provided by Hyrax, and only interacts with the service through the interfaces provided by Hadoop. The application code simply views the cloud as one computation and storage unit, when in reality it is much more complicated than that. It would have required a lot more work to implement

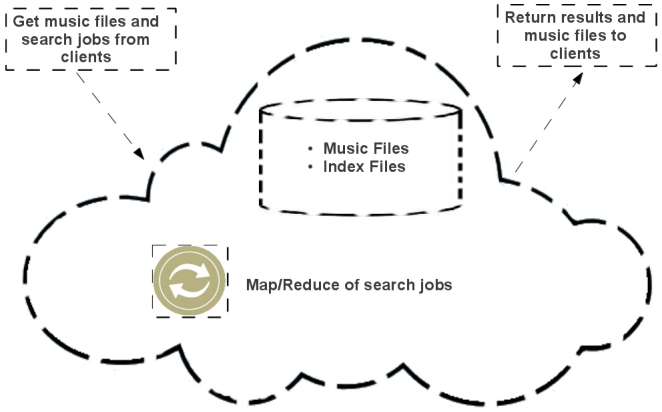

Figure 6.2: Design of cloud in *MusicDJ*. This cloud is virtual, and is the view that each client / mobile node sees

MusicDJ without the benefits of using Hyrax.

On the front end, once the user selects the local folder to upload to the HDFS, our Android application automatically goes through all the music files within the folder and extracts their metadata (ie title, artist, length), compiles the metadata into a index file, and uploads this as well as the actual music files to the HDFS through the interface provided by Hadoop.

Besides the mobile devices, *MusicDJ* does not require any other hardware or software except for a machine to act as the NameNode and JobTracker. No dedicated server for the application is required as all the upload, search and download functionalities are included in the Android application itself.

Note that we did not make use of the dynamic class loading functionality that is provided with Hyrax (described in Subsection 4.2.1) as every node already has the code required for the MapReduce job. It therefore makes no sense to have to transfer the code to the nodes when executing jobs as this would only increase communications cost (and hence battery consumption). However, if another device had only the Hyrax backend without the application code of *MusicDJ* itself, dynamic class loading would enable that device to act as a node within the cluster as well even though the device is not a client of *MusicDJ*.

MusicDJ is currently only able to handle MP3 music files, but this can easily be extended as the Android operating system is capable of handling many different types of multimedia files. We chose to use MP3 files as we already possess a significant collection of these files in our own personal collection that we can use to test and evaluate *MusicDJ*.

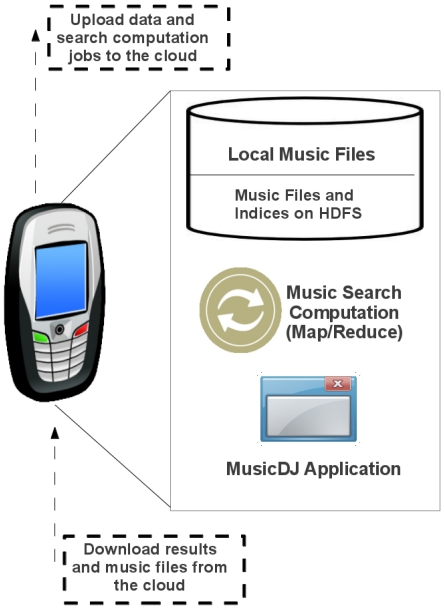

Figure 6.3: Design of mobile node in *MusicDJ*. The mobile node is the location of actual computation and storage of data that logically belongs to the cloud. It sees the cloud as a service and uploads data and computation to it.

A screenshot of *MusicDJ* is shown in Figure 6.4.

6.4 Evaluation

Similar to our testbed in Chapter 5, we tested *MusicDJ* on a testbed of Nexus S devices running Android 4.1.1. We used the same router and same machine as the master node as that mentioned in Chapter 5. We ran *MusicDJ* on a cluster of $3, 5, 7, 10$ and 12 phones in order to determine how well the search functionality that was implemented using the MapReduce framework scaled with the cluster size. We format and restart the cluster for each cluster size we run the evaluations for so that the evaluations always start in the same state.

In our evaluation, only one device is performing an operation (either upload or search) at any one time. While this is not realistic since it is more likely that multiple users will be accessing the system simultaneously in a real-life setting, it gives a good indication of the

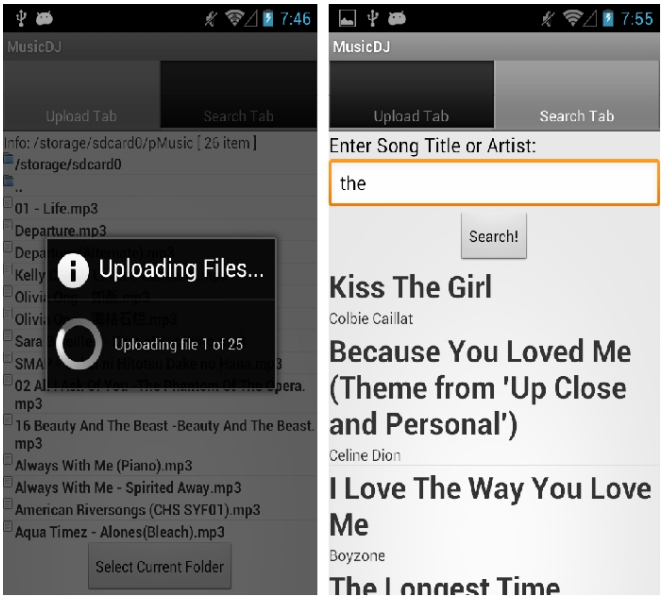

Figure 6.4: Screenshots of the MusicDJ application. On the left is a view of an upload in progree, on the right is a view of search results

baseline performance of *MusicDJ*. Unfortunately, emulating the real-world usage patterns of users is difficult to achieve. We therefore do not attempt to do so.

We evaluate *MusicDJ* by the time taken to upload music files to the cloud and to perform a search for music files. We focus on these two operations as they are the most time-consuming ones and are the ones directly related to Hyrax.

6.4.1 Test Data

We used a folder of 25 MP3 files as our test set for *MusicDJ*. The total size of the folder is 119.2 MB. We used the same set of files for every device.

6.4.2 Results

File Upload

Since the replication factor was set at 2, we do not anticipate significant differences in file upload times for the different cluster sizes as the length of the write pipeline is the same for each case.

To determine file upload times, we upload the same set of music files mentioned in Subsection 6.4.1 from each device in each case and take the average of the upload times. We measure search time from the time the user initiates the search till the results are returned to the user for display. Our findings are displayed in Figure 6.5. We also provide an estimate of the average upload throughput in Table 6.1.

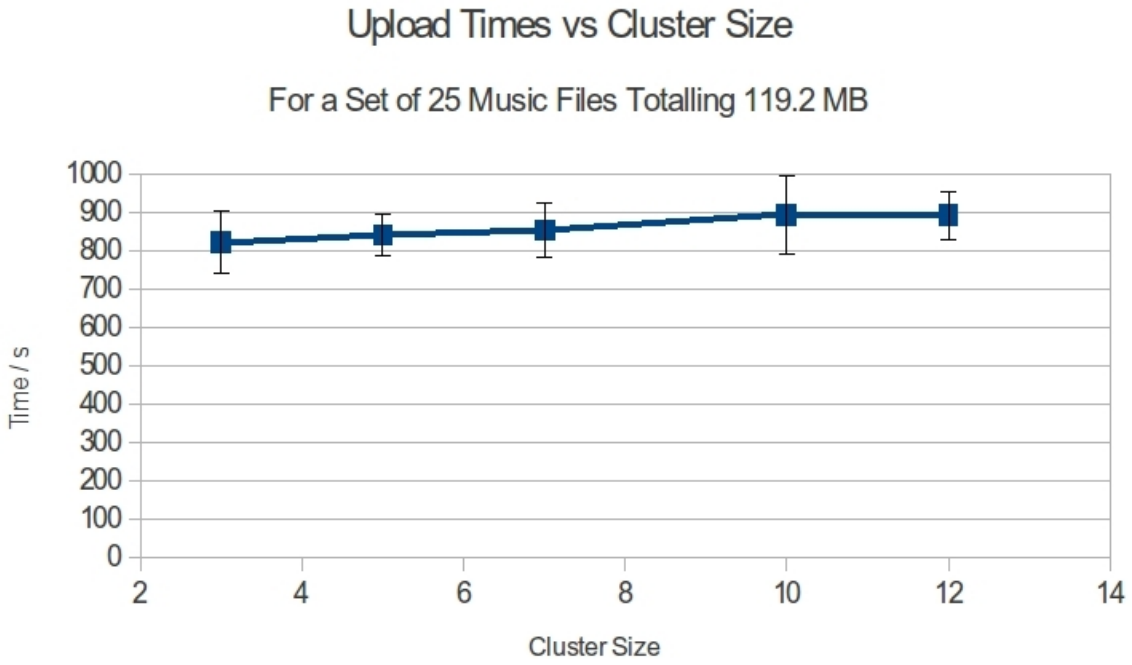

Figure 6.5: Graph of upload times against cluster size with standard deviation included

Figure 6.5 and Table 6.1 both show that the upload times and speed were not significantly affected by the cluster size. This agrees with our expectations since the pipeline length was always the same during data write as the replication factor was kept constant. However, the upload times were approximately 15 to 16 minutes long in each case. Such

Cluster Size	3 Nodes	5 Nodes	7 Nodes	10 Nodes	12 Nodes
Average throughput (MB/s)	0.14	0.14	0.14	0.13	0.13

Table 6.1: Average upload throughput for different cluster sizes

long upload times for only 25 music files might be unacceptable to users who are unwilling to wait.

In contrast, uploading the same set of files onto DropBox [21], which is a file hosting service, over the Internet through a wireless connection (not 3G) from a Nexus S took approximately 210 seconds, which translates to a throughput of approximately 0.57 MB/s. The relatively huge disparity between this and the upload times observed for *MusicDJ* could be due to the slower write rates on the mobile devices, as well as inefficiencies within HDFS itself.

One possible way to reduce file upload times is to reduce the replication factor. For example, when uploading the same set of files with only one node in the cluster, the upload took approximately 461 seconds. However, this involves a tradeoff with file availability, as the risk of a file being lost from the cluster due to node departures or failures is higher.

File Search

We evaluate file search times by first uploading the test files from each device onto the cloud. Although the test files uploaded by each device is the same, our system will still perform a full search on every index file found so that on average, each node on the cluster should search through the same amount of data. We therefore do not anticipate significant differences in the seach times for the different cluster sizes as well.

We determine file search times by performing the search on each device in the cluster and taking the average of the search times. We vary the search terms on each device in order to inject some variety in the search jobs. Our findings are displayed in Figure 6.6.

Figure 6.6 shows that the average search times for a file increases with the cluster size, but it appears to plateau as the cluster size increases further. The increase in time is possibly due to the increased overhead in setting up the search jobs to run on more nodes, which requires more data transfer and control messages. The graph does seem to suggest, however, that with more nodes, the cluster will be able to scale well with respect to the job time.

We believe that the benefits of Hyrax on file search times will emerge with more nodes

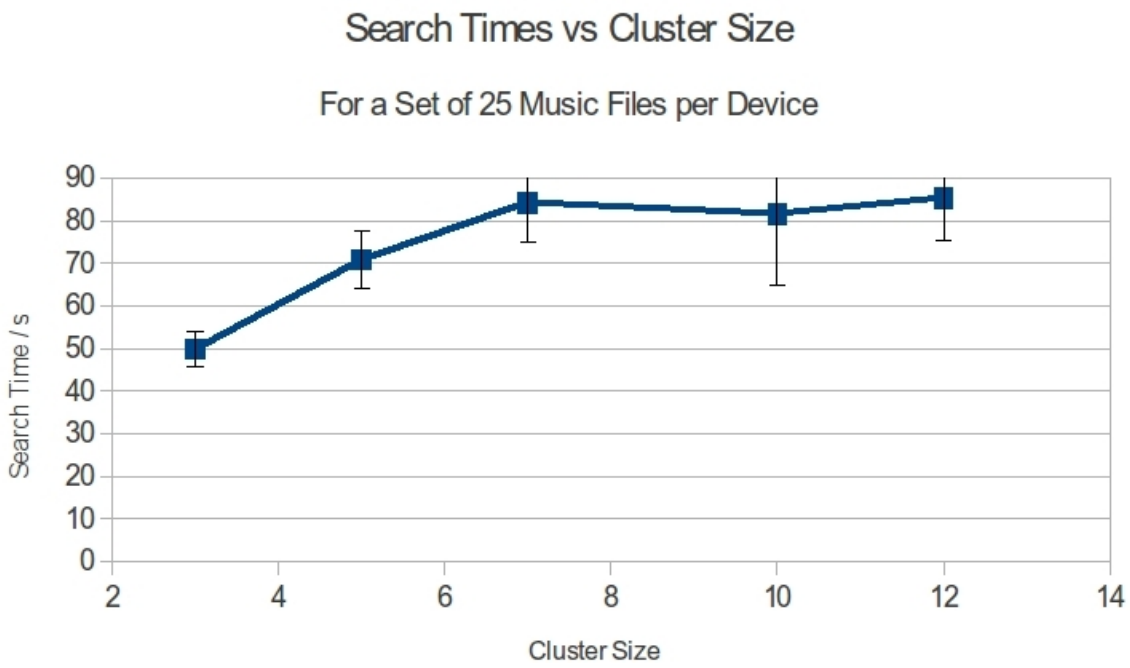

Figure 6.6: Graph of search times against cluster size with standard deviation included

and more data to search through. If the search were to be performed on one single node alone (ie without using Hyrax), not only would an index of available music files from other nodes need to be downloaded from each individual node, but the volume of information that the node needs to search through would increase by a factor of the cluster size as well. Ideally, assuming it takes time d to download the index from another node, and time t to execute the search job on just a single node alone, Hyrax would be beneficial if

$$\frac{t}{n} + s < t + dn$$

, where n is the cluster size, and s is the time required to set up the search job on Hyrax (we assume s is not affected by n). That is, we require $s < t\left(\frac{n-1}{n}\right) + dn$. This would be easier to achieve with a higher value of n. Therefore, the more nodes we have in the cluster, the more benefits users will be able to derive from Hyrax.

Chapter 7

Conclusions

The full potential of both cloud computing and mobile applications have not been realised yet. We believe implementing a cloud computing infrastructure on mobile devices is a step towards realising that potential. Hyrax provides such an infrstructure that allows for the mobile devices to be storage and computation nodes within a cloud computing cluster, hence creating a local mobile cloud.

Hyrax is based on Hadoop. We feel that Hadoop was a natural choice for Hyrax, since it was already built with certain assumptions in mind that apply in the case of a cluster made up of mobile devices. Such assumptions include node departures and hardware failure, both of which are handled automatically by Hadoop.

While basing Hyrax on Hadoop has its benefits, there are disadvantages as well. For example, Hyrax has a relatively high overhead cost, which is magnified when resource-constrained mobile devices are used [33].

Nevertheless, Hyrax provides a convenient and abstract interface for implementing distributed applications on mobile devices. We demonstrated this through our music sharing application, where the code of the actual search function was written fairly quickly (in a much shorter time than the code for the user interface).

We describe some of the challenges we faced in the development of Hyrax in Section 7.1, and some possible future work in Section 7.2.

7.1 Challenges

7.1.1 Android Fragmentation

While the choice of developing Hyrax for the Android system provided numerous benefits (see Section 3.5, the open nature of the Android system was also the source of some development challenges. One such prime example is that of fragmentation. Fragmentation refers to the numerous devices and versions (including manufacturer versions) of the Android OS available on the market. [36] gives a typical scenerio where developers have to test their applications on as many devices running Android as possible in order to be sure that their applications will work as expected for users. However, this provides no guarantee that the application will run on any untested device. In fact, [36] mentions that complaints about applications not supporting their devices is one of the most common complaints on the Android Market (also known as Google Play since 6 March 2012 [28]).

We also faced similar problems during the development of Hyrax. Hyrax was originally developed and tested on an Android G1 [30] running Android 2.1 (Update 1). We subsequently managed to procure some Nexus S devices [27] running Android 2.3. However, running Hyrax on the Nexus S exposed certain behavioural differences in the code that had to be fixed. The version of Android for the Nexus S was subsequently upgraded to version 4.1.1 (released 9 July 2012), which led to further necessary modifications to make Hyrax work on the new version of Android. We anticipate that running Hyrax off other devices or subsequent versions of the Android operating system will probably require changes to the Hyrax code again as further problems due to behavioural differences are exposed.

7.1.2 Size of Hadoop Codebase

An unexpected challenge that manifested itself early on in the development of Hyrax is the sheer size of the Hadoop codebase. For example, a compressed version of Hadoop 0.21.0 is approxmiately 71 MB in size [6]. The Eclipse Integrated Development Environment (IDE) [22], which is usually used for Android application development, could not handle the compilation of this large amount of code on our development machine without running out of memory. Even when we compiled from the shell using `ant` [4], each run of the compile process took at least 2 minutes. This resulted in a lot of wasted time during development, especially when only a few lines of code were changed each time. It also robbed us of the benefits of using an IDE to develop our code.

7.1.3 Debugging

Debugging Hyrax also proved to be a challenge. Due to the size of Hyrax, it was prohibitively slow to run it on a standard Android emulator or with any form of a debugger. All debugging was hence done on runs on actual devices. In order to see what was going on, it became necessary to study the Hyrax logs, as well as use the Android `logcat` utility, to read log messages and any error messages to determine if any errors or unexpected behaviours have occured. This was time consuming, and required attention as error messages tend to scroll past the top of the screen relatively quickly. It sometimes became necessary to add more debugging messages into the Hyrax code in order to determine the exact location where an error or unexpected behaviour occured. All these served to increase development time as adding more debugging messages meant having to recompile Hyrax.

7.2 Future work

7.2.1 Hyrax on other Mobile Platforms

Hyrax currently only works on the Android platform. While this enables a substantial proportion of the mobile users population to make use of Hyrax, it still leaves out a significant proportion of users of other platforms such as the iPhone or BlackBerry. It would be beneficial to develop Hyrax for these other platforms as well so that we can create mobile clouds that are based not only on different mobile devices, but different mobile operating systems and platforms as well. In this way, Hyrax would truly be working on a heterogeneous cloud.

7.2.2 Improving Hadoop Performance

Bahl et al [8] advocate performing optimisations on current cloud infrastructure to specially cater to the needs and requirements of mobile applications. They cited the example of zCloud [41], which is a cloud computing infrastructure developed by Zynga specially for their popular mobile games.

Previous research and studies have shown that Hadoop suffers from performance issues. For example, [32] focuses on making Hadoop more energy efficient, while other studies have focused on improving Hadoop's node failure recovery performance (for ex-

ample, see [19]). It may be worthwhile to look at these studies and implement their methods to Hyrax in order to achieve similar improvements in the performance of Hyrax.

Furthermore, as mentioned in Subsection 5.3.3, an investigation into any inefficiencies in either the HDFS or MapReduce framework for Hadoop might need to be investigated as the amount of network traffic does not correspond to the amount of input data being processed by Hyrax.

7.2.3 Reducing Power Consumption

As noted in Section 1.1, battery life is still of topmost concern to mobile users. Therefore, it is important to manage the power consumption of Hyrax in order to attract users to actually use Hyrax.

Further studies on the power consumption of Hyrax could be undertaken in the future. For example, studies on the power consumption by different tasks on Hyrax could be performed. However, it is not clear if such studies might turn out to be useful as we feel that power consumption by tasks is highly dependent on individual applications. Other studies taken to improve the power consumption of Hadoop itself could also be explored and possibly implemented in Hyrax.

7.2.4 Switching Clusters

We envision that users will switch clusters often as they move about and change location, coming into range of different clusters all the time. On the backend, this requires the datanode to clean up all the data blocks that were already stored on the device from a previous cluster in order to be able to accept blocks from a new cluster. Another, perhaps more useful, feature would be for the datanode to be able to store data blocks from different clusters on the device, and only access those blocks that belong to the cluster that the device is currently connected to. Hyrax currently only supports the use of one cluster, and requires manual intervention to clean up the blocks stored on the nodes before they can connect to a new cluster. Support for such a feature could be implemented on Hyrax in the future.

7.2.5 Mobile Rack-awareness

Hyrax, unlike Hadoop, currently has no concept of a "rack". Rack information is used by Hadoop in order to determine block placement locations. Hadoop assumes that nodes on the same rack were located in the same location physically. Thus, nodes within the same rack could communicate with each other easily, but could also fail together in the case of rack failure.

To apply the "rack" concept to our mobile cloud, we could assign mobile devices connected to the network at the same point (for example, perhaps through the same router) to the same "rack". Such mobile devices would experience shorter latencies since communications between them would only require one hop through a router. On the other hand, should the router fail, both nodes could possibly be lost as well unless they manage to connect to another router or network access point. Given that the physical location of the nodes can change, our system will have to be able to reassign "racks" on the fly as the mobile devices switch wireless access points. This would further affect the block replication strategy. A detailed study will have to be performed to determine the feasibility of such a scheme, or if a better scheme could be devised.

7.2.6 Adaptive Replication and Selection of Active Nodes

Work has been done to explore the effects that the replication factor for files has on energy consumption in a Hadoop cluster [32]. A study could be undertaken to see if the work and suggestions mentioned in [32] could be adapted for use in Hyrax. For example, suggestions on putting a portion of the nodes into a less power-consuming standby mode could be adopted so that communications between these nodes and the master node need not happen too frequently, which will save power due to reduced network traffic. This is, however, at a tradeoff with data availability and node response time.

7.2.7 Security

The data stored on the HDFS for Hyrax is physically stored on numerous mobile devices, each with a different owner. It is possible for the owners of each device to read the data blocks stored on their own device. We currently envision that users should be able to read all files within the HDFS, since the HDFS is a shared storage space. However, should there be a future need to restrict file access, schemes such as data encryption could be employed in order to protect the data blocks from being read by unauthorised users.

Another potential issue is that the data blocks are currently stored within the external stroage space on each device. By design, Android allows any application and user to access this storage space. This means that users can potentially modify or remove the data blocks stored locally on their device. Studies should be done to understand the effects of any such events, and whether the NameNode is able to recover from such events and replicate the affected blocks. Another possible solution is to look into storing data blocks in a part of the Android filesystem which only allows for restricted access.

7.2.8 Optimisation or re-implementation of MapReduce

Given the high overheard required to run Hadoop, it may be more beneficial to create a new implementation of the MapReduce framework that is more mobile-friendly. Such an implementation should be created with the resource limitations and other aspects of mobile devices in mind. Such limitations include memory limitations, power limitations and processing capabilities limitations. It should be possible to optimise MapReduce to not just use resources more efficiently, but it may also be possible for to re-implement MapReduce with a smaller code base that will not only reduce application size, but also be easier to maintain.

7.2.9 Large-scale Testing

Hyrax has only been tested on a relatively small testbed of up to 12 devices. Given that Hyrax was designed to scale to hundreds or even thousands of devices, further tests should be conducted to verify that it is indeed capable of scaling to such large numbers. It will be difficult to procure so many devices for the sole purpose of testing, so asking for volunteer participation from existing mobile users will probably be required. Not only will this keep cost down, but this will also allow Hyrax to be tested in a more realistic setting since the mobile devices will also be used for daily normal use. This provides for the possibility of studying how using Hyrax might affect the day-to-day operations of the mobile devices.

7.2.10 Offloaded vs. Local Computation

Depending on the application, it might sometimes be more beneficial to offload only part of the computation to the Hyrax infrastructure and perform the remaining computation locally. This, however, requires a study on the tradeoffs between local computation and offloading. Such a study could possibly be undertaken in the future so that the perfor-

mance of Hyrax and distributed jobs on mobile devices can be improved both in terms of execution time and resource usage.

Bibliography

[1] ALLIANCE, O. H. Android. http://www.android.com, 2008. [Online; accessed 17 July 2012]. 1.1.5

[2] AMAZON. Amazon elastic compute cloud (amazon ec2). http://aws.amazon.com/ec2/, 2006. [Online; accessed 21 July 2012]. 1.1.2

[3] ANGIN, P., BHARGAVE, B., AND HELAL, S. A mobile-cloud collaborative traffic lights detector for blind navigation. In *2010 Eleventh International Conference on Mobile Data Management* (May 2010), pp. 369 – 401. 1.1.3, 2.1

[4] APACHE. Apache ant. http://ant.apache.org, 2000. [Online; accessed 8 August 2012]. 7.1.2

[5] APACHE. Hadoop. http://hadoop.apache.org, 2007. [Online; accessed 16 July 2012]. 1.1.4

[6] APACHE. Hadoop 0.21.0. http://apache.cs.utah.edu/hadoop/common/hadoop-0.21.0/, 2010. [Online; accessed 8 August 2012]. 7.1.2

[7] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A., STOICA, I., AND ZAHARIA, M. Above the clouds: A berkeley view of cloud computing. Tech. rep., University of California, Berkeley, February 2009. 1.1.2

[8] BAHL, P., HAN, R. Y., LI, L. E., AND SATYANARAYANAN, M. Advancing the state of mobile cloud computing. In *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services* (New York, NY, USA, 2012), MCS '12, ACM, pp. 21 – 28. 2, 7.2.2

[9] BALASUBRAMANIAN, N., BALASUBRAMANIAN, A., AND VENKATARAMANI, A. Energy consumption in mobile phones: A measurement study and implications for

network applications. In *Proceedings of the 9th ACM SIGCOM conference on Internet measurement conference* (New York, NY, USA, 2009), IMC '09, ACM, pp. 280 – 293. 5.5.4

[10] BAO, X., AND ROY CHOUDHURY, R. Movi: Mobile phone based video highlights via collaborative sensing. In *Proceedings of the 8th International Conference on Mobile Systems, Applications and Services* (New York, NY, USA, 2010), MobiSys '10, ACM, pp. 357 – 370. 2.4

[11] BLACK, M., AND EDGAR, W. Exploring mobile devices as grid resources: Using an x86 virtual machine to run boinc on an iphone. In *2009 10th IEEE/ACM International Conference on Grid Computing* (October 2009), pp. 9 – 16. 1.1.3

[12] BORTHAKUR, D., GRAY, J., SARMA, J. S., MUTHUKKARUPPAN, K., SPIEGELBERG, N., KUANG, H., RANGANATHAN, K., MOLKOV, D., MENON, A., RASH, S., SCHMIDT, R., AND AIYER, A. Apache hadoop goes realtime at facebook. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2011), SIGMOD '11, ACM, pp. 1071 – 1080. 3.4, 4.2.2

[13] CARROLL, A., AND HEISER, G. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2010), USENIXATC'10, USENIX Association, pp. 21 – 21. 5.5.3

[14] CHUN, B.-G., IHM, S., MANIATIS, P., NAIK, M., AND PATTI, A. Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems* (New York, NY, USA, 2011), EuroSys '11, ACM, pp. 301 – 314. 2.3

[15] CHUN, B.-G., AND MANIATIS, P. Augmented smartphone applications through clone cloud execution. In *Proceedings of the 12th Conference on Hot Topics in Operating Systems* (Berkeley, CA, USA, 2009), HotOS'09, USENIX Association, pp. 8 – 8. 2, 2.3

[16] COMSCORE. Exponential tablet adoption in 2011 ushers in era of convergent consumption. `http://www.comscoredatamine.com/2012/03/exponential-tablet-adoption-in-2011-ushers-in-era-of-convergent-consumption/`, March 2012. [Online; accessed 9 August 2012]. 1.1.1

[17] CRACIUNAS, S. S., HAAS, A., KIRSCH, C. M., PAYER, H., RÖCK, H., ROTTMANN, A., SOKOLOVA, A., TRUMMER, R., LOVE, J., AND SENGUPTA, R. Information-acquisition-as-a-service for cyber-physical cloud computing. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing* (Berkeley, CA, USA, 2010), HotCloud'10, USENIX Association, pp. 14 – 14. 2.4

[18] DEAN, J., AND GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. *Commun. ACM 51*, 1 (January 2008), 107 – 113. 1.1.4

[19] DINU, F., AND NG, T. E. Understanding the effects and implications of compute node related failures in hadoop. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing* (New York, NY, USA, 2012), HPDC '12, ACM, pp. 187 – 198. 7.2.2

[20] DOU, A., KALOGERAKI, V., GUNOPULOS, D., MIELIKAINEN, T., AND TUULOS, V. H. Misco: A mapreduce framework for mobile systems. In *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments* (New York, NY, USA, 2010), PETRA '10, ACM, pp. 32:1 – 32:8. 1.1.3, 2.2.1

[21] DROPBOX. Dropbox: Simplify your life. `http://www.dropbox.com`, 2008. [Online; accessed 30 July 2012]. 5.4.2, 6.4.2

[22] ECLIPSE. The eclipse foundation open source community website. `http://www.eclipse.org`, 2004. [Online; accessed 8 August 2012]. 7.1.2

[23] ELESPURU, P. R., SHAKYA, S., AND MISHRA, S. Mapreduce system over heterogeneous mobile devices. In *Proceedings of the 7th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems* (Berlin, Germany, 2009), SEUS '09, Springer-Verlag, pp. 168 – 179. 2.2.1

[24] ERICSSON. Traffic and market report. [Online; accessed 18 July 2012]. 1.1.1

[25] GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S.-T. The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2003), SOSP '03, ACM, pp. 29 – 43. 1.1.4

[26] GIURGIU, I., RIVA, O., JURIC, D., KRIVULEV, I., AND ALONSO, G. Calling the cloud: Enabling mobile phones as interfaces to cloud application. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware* (New York, NY, USA, 2009), Middleware '09, Springer-Verlag New York, Inc., pp. 5:1 – 5:20. 2.3

[27] GOOGLE. Google nexus s technical specifications. http://www.google.com/nexus/s/tech-specs.html, 2010. [Online; accessed 16 July 2012]. 5.1.1, 7.1.1

[28] GOOGLE. Google play. https://play.google.com/, 2012. [Online; accessed 1 August 2012]. 7.1.1

[29] GUAN, L., KE, X., SONG, M., AND SONG, J. A survey of research on mobile cloud computing. In *Proceedings of the 2011 10th IEEE/ACIS International Conference on Computer and Information Science* (Washington, DC, USA, 2011), ICIS '11, IEEE Computer Society, pp. 387 – 392. 2

[30] HTC. Htc dream specifications. http://www.htc.com/www/product/dream/specification.html, 2008. [Online; accessed 30 December 2010]. 7.1.1

[31] KUMAR, K., AND LU, Y.-H. Cloud computing for mobile users: Can offloading computation save energy? *Computer 43*, 4 (April 2010), 51 – 56. 1.1.3

[32] LEVERICH, J., AND KOZYRAKIS, C. On the energy (in)efficiency of hadoop clusters. *SIGOPS Oper. Syst. Rev. 44*, 1 (March 2010), 61 – 65. 3.3.2, 7.2.2, 7.2.6

[33] MARINELLI, E. Hyrax: Cloud computing on mobile devices using mapreduce. Tech. Rep. CMU-CS-09-164, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, September 2009. 1.2, 7

[34] MIETTINEN, A. P., AND NURMINEN, J. K. Enerty efficiency of mobile clients in cloud computing. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing* (Berkeley, CA, USA, 2010), HotCloud'10, USENIX Association, pp. 4 – 4. 1.1.3

[35] MYERS, A. High availability for the hadoop distributed file system (hdfs). http://www.cloudera.com/blog/2012/03/high-availability-for-the-hadoop-distributed-file-system-hdfs/, 2012. [Online; accessed 22 July 2012]. 3.3.2

[36] PANZARINO, M. The shocking toll of hardware and software fragmentation on android development. http://thenextweb.com/mobile/2012/03/30/the-shocking-toll-of-hardware-and-software-fragmentation-on-android-development/, March 2012. [Online; accessed 1 August 2012]. 7.1.1

[37] POWER, J. D., AND ASSOCIATES. Smartphone battery life has become a significant drain on customer satisfaction and loyalty. *Wireless Smartphone and Traditional Mobile Phone Satisfaction Studies 1* (2012). 1.1.1

[38] SATYANARAYANAN, M., BAHL, P., CACERES, R., AND DAVIES, N. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE 8*, 4 (Oct – Dec 2009), 14 – 23. 2.1, 2.2

[39] SHVACHKO, K., KUANG, H., RADIA, S., AND CHANSLER, R. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* (Washington, DC, USA, 2010), MSST '10, IEEE Computer Society, pp. 1 – 10. 1.1.4

[40] VERBELEN, T., SIMOENS, P., DE TRUCK, F., AND DHOEDT, B. Cloudlets: Bringing the cloud to the mobile user. In *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services* (New York, NY, USA, 2012), MCS '12, ACM, pp. 29 – 36. 2.2

[41] ZYNGA, I. The evolution of zcloud. `http://code.zynga.com/2012/02/the-evolution-of-zcloud`, 2012. [Online; accessed 31 July 2012]. 7.2.2