Hyrax: Cloud Computing on Mobile Devices using MapReduce

Eugene E. Marinelli

CMU-CS-09-164

September 2009

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Priya Narasimhan, Chair
Srinivasan Seshan

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

Copyright © 2009 Eugene E. Marinelli

This work has been partially supported by the National Science Foundation CAREER grant CCR-0238381 and CNS-0326453, and the Army Research Ofce grant number DAAD19-02-1-0389 (Perpetually Available and Secure Information Systems) to the Center for Computer and Communications Security at Carnegie Mellon University.

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 the U.S. Government.

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

Abstract

Today's smartphones operate independently of each other, using only local computing, sensing, networking, and storage capabilities and functions provided by remote Internet services. It is generally difficult or expensive for one smartphone to share data and computing resources with another. Data is shared through centralized services, requiring expensive uploads and downloads that strain wireless data networks. Collaborative computing is only achieved using ad hoc approaches.

Coordinating smartphone data and computing would allow mobile applications to utilize the capabilities of an entire smartphone cloud while avoiding global network bottlenecks. In many cases, processing mobile data in-place and transferring it directly between smartphones would be more efficient and less susceptible to network limitations than offloading data and processing to remote servers.

We have developed Hyrax, a platform derived from Hadoop that supports cloud computing on Android smartphones. Hyrax allows client applications to conveniently utilize data and execute computing jobs on networks of smartphones and heterogeneous networks of phones and servers. By scaling with the number of devices and tolerating node departure, Hyrax allows applications to use distributed resources abstractly, oblivious to the physical nature of the cloud.

The design and implementation of Hyrax is described, including experiences in porting Hadoop to the Android platform and the design of mobile-specific customizations. The scalability of Hyrax is evaluated experimentally and compared to that of Hadoop. Although the performance of Hyrax is poor for CPU-bound tasks, it is shown to tolerate node-departure and offer reasonable performance in data sharing. A distributed multimedia search and sharing application is implemented to qualitatively evaluate Hyrax from an application development perspective.

Acknowledgments

I would like to acknowledge the many people who supported me in completing this work.

First, I would like to acknowledge Professor Priya Narasimhan, my advisor, for her guidance, inspiration, and funding over the past two years. I would also like to thank Professor Srinivasan Seshan for serving on my thesis committee.

I would like to thank Jiaqi Tan for his constant willingness to contribute and discuss ideas for my project. The Hadoop log analysis system that he developed has been instrumental in debugging and evaluating Hyrax. I would also like to thank Dr. Rajeev Gandhi and Ernie Brown for taking the time to help me with testing at Mellon Arena. I would also like to acknowledge the rest of the people with whom I collaborated in research and class projects this year, including Mike Kasick, Keith Bare, Soila Kavulya, Austin McDonald, Dmitriy Ryaboy, Nathan Mickulicz, and Shahriyar Amini.

I would like to thank Jennifer Engleson for all her help in acquiring hardware for this project, answering various questions, and, most importantly, notifying me when free food was available. I would also like to thank Deborah Cavlovich for her help in the Fifth Year Master's program, and Samantha Stevick and Joan Digney for their help with technical reports and posters. I would like to thank Karen Lindenfelser for all she has done to create a great work environment at the CIC, which was practically my home this year.

I would like to thank my office-mates Adam Goldhammer, Michael Chuang, Heer Gandhi, Wesley Jin, and James Kong for helping me formulate and refine my ideas throughout the project. Adam's and Heer's hardware knowledge was particularly helpful. I would also like to thank my apartment-mates Rich Lane and Jeffrey Ohlstein for giving me feedback on my ideas and providing technical advice. I would like to thank Jessica Liao for helping to revise my thesis and prepare for my defense, and for supporting me in general throughout the year.

I would like to thank Seth Goldstein for recommending me for the Fifth Year Master's program and for his guidance and advice during my time as a student at Carnegie Mellon.

I would also like to thank Frank Pfenning for hiring me as a teaching assistant and advising me on research and academics.

Finally, I would like to thank my father, Gene Marinelli, my mother, Barbara Marinelli, my sister, Amy Marinelli, and the rest of my family and friends for their support this year and throughout my education.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Most of today's smartphone applications are geared towards an individual user and only use the resources of a single phone. There is an opportunity to harness the collective sensing, storage, and computational capabilities of multiple networked phones to create a distributed infrastructure that can support a wealth of new applications. These computational resources and data are largely underutilized in today's mobile applications. Using these resources, applications could conveniently use the combined data and computational abilities of an entire network of smartphones to generate useful results for clients both outside and within the mobile network. This interface and the underlying hardware would create a *mobile-cloud* upon which compute jobs could be performed. We define *mobile-cloud computing* to be an extension of cloud computing in which the foundational hardware consists at least partially of mobile devices.

Some mobile applications already extract and aggregate information from multiple phones. Tweetie Atebits for the iPhone uses locations from other phones running the application to allow users to see recent Twitter posts by nearby users. Video and photo publishing applications such as YouTube and Flickr allow users to upload multimedia data to share online. The Ocarina application Smule for the iPhone allows users to listen to songs played by other users of the application, displaying the location of each user on a globe. Such smartphone applications are "push"-based and centralized, meaning that users push their information to a remote server where it is processed and shared.

It is possible to use a networked collection of smartphones in a more opportunistic way. Each smartphone has some amount of storage, some amount of compute power, some sensing abilities, some multimedia data, and some amount of energy. Each of these capabilities is currently only available to and utilized by the smartphone's owner. What if these capabilities were somehow offered to other users and applications? What if we could harness a collection of smartphones to support large-scale distributed applications, using smartphones as the basis for a cloud computing infrastructure? Each smartphone would be equipped to perform individual, local computations on its local data in support of a larger, system-wide objective, and the outcomes of each smartphone's local actions would be aggregated to meet the needs of the overall application. Applications could use these resources abstractly, oblivious to the underlying implementation on a smartphone network.

Similar concepts have been studied in sensor networks and mobile grid computing, Sorniotti et al. [2007], Akyildiz et al. [2006], Litke et al. [2004]. However, in contrast to data-center "pay-as-you-use" cloud computing and sensor networks, in the proposed mobile-cloud concept (1) each node is owned by a different user, (2) each node is likely to be mobile, (3) the network topology is more dynamic, and (4) each mobile node is battery-powered. In contrast to mobile grid computing, mobile-cloud computing focuses on abstracting away from the implementation of resource sharing to provide a useful tool for applications.

Using mobile hardware for cloud computing offers advantages over using traditional hardware, such as computational access to multimedia and sensor data without large network transfers, more efficient access to data stored on other mobile devices, and distributed ownership and maintenance of hardware. Such a concept inevitably gives rise to many concerns, including access-control, incentivisation of users, privacy, and mobile resource conservation. At the same time, this concept may create many opportunities for interesting new applications and for more resource-efficient versions of existing applications.

One application that illustrates the usefulness of a mobile-cloud computing platform is distributed mobile multimedia sharing. Today, it is easy to upload mobile photos and videos directly to remote services such as Flickr and YouTube, at least when a stable, high-bandwidth network connection is available. However, sharing a file in this way is expensive and sometimes wasteful. The file needs to be compressed, annotated, and then sent over the network, draining the battery in the process. This upload is also a burden for wireless network service providers who must handle these large uploads and for other mobile users who experience the network performance degradation that results. Furthermore, many videos and pictures uploaded to these websites are only accessed a few times if at all.

Handling file uploads is a particularly big problem when a large number of people are using mobile phones in one location. For example, many wireless data services failed in Washington D.C. during the 2009 U.S. Presidential Inaugration Park [2009], an event which millions of people attended. In fact, CTIA-The Wireless Association issued a press release Joe Farren before the event imploring mobile users to "wait until leaving the Inaugural events to send [photos and videos] to friends and family". In such cases, mobile users are unable to publish and consume multimedia at the time when it is most interesting and relevant. Furthermore, wireless service providers cannot prepare for all events that induce high network traffic in advance. For instance, the terrorist attacks of September 11th, 2001, induced similar network congestion in New York Beard [2005], making it difficult to place wireless phone calls. If a similar incident happened today, it would be difficult to share extremely important multimedia data collected on smartphones.

A more scalable way to support multimedia sharing from mobile devices is to host files on phones and distribute queries and summarization tasks across many nodes, eliminating the need for each user to upload large files to and retrieve files from remote services. Instead, large transfers could be performed directly within local networks. Search queries could include times and locations of interest and incorporate local sensor readings such as heading and movement to estimate video quality and relevance. Irrelevant and low-quality videos would never need to leave the phone on which they were collected, saving battery energy for both these users and users who would have downloaded these videos, and reducing the load on the network as a whole. Data "hot-spots", i.e. smartphones uniquely hosting very popular data, could be avoided by replicating popular data to other smartphones, and in some cases servers on the local network. Using this system, smartphone users could publish and retrieve photos and video from many vantage points without waiting until after the event, and other entities such as broadcasters could find relevant videos to share with the general public. This application would be useful at any event where a large crowd is gathered, such as sporting events, concerts, plays, and movies.

1.1 Our contributions

The goal of our research is to develop a *mobile-cloud infrastructure that will enable smartphone applications that are distributed both in terms of data and computation*. In this paper, we present our implementation and evaluation of a mobile-cloud computing infrastructure based on MapReduce.

We needed a starting point for our investigation of mobile-cloud computing. One possibility was to build a new infrastructure from scratch designed to run on mobile devices. Instead, we decided to start with an existing cloud computing infrastructure and examine its suitability by modifying it to run on mobile devices. We sought to understand and articulate the obstacles, challenges, and solutions in supporting a mobile-cloud computing platform.

We started with Hadoop Apache, an open-source implementation of MapReduce Dean and Ghemawat [2008]. MapReduce is a programming framework and implementation introduced by Google for data-intensive cloud computing on commodity clusters. Hadoop is used by companies such as Yahoo!, Facebook, and IBM Apache to process large amounts of data distributed across a network of servers. It is commonly used on Amazon's Elastic Compute Cloud (EC2), a utility computing service.

Hadoop includes a large amount of the functionality required for a mobile-cloud computing system. We target the Android platform since it incorporates the Dalvik Java VM, which is capable of executing much of Hadoop's Java codebase without modification. We provide an overview of cloud computing, Hadoop, and Android in §2.

In this paper, we establish the motivation for this mobile-cloud computing platform, which we call Hyrax [1], discuss how the challenges of mobile computing apply to this platform, enumerate the requirements of the platform, and describe the choices we made and the challenges we faced in porting Hadoop to run on Android. We develop and present the results of several experiments that evaluate the scalability, flexibility, performance, and battery usage of Hyrax. We also implement a distributed multimedia search and sharing application to gain insight into the advantages of using Hyrax as an infrastructure for applications that use mobile data.

Our experiments show that Hyrax easily scales to the 12 Android smartphones in our testbed in terms of execution times and resource usage. Unfortunately, it also exerts a huge base cost on Android, requiring a lot of time to process relatively small amounts of data. Hyrax handles node-departure during MapReduce jobs when the number of nodes in the cluster and the replication factor are sufficiently high. Hyrax also allows for data sharing among smartphones in a WiFi network with similar but potentially more consistent latencies compared to uploading files to and hosting files from a remote server. Hyrax has not been optimized to be battery-efficient, but it uses significantly less power than video recording and downloading even in the worst case.

This document is organized as follows: §2 provides the necessary background on current smartphone technology, cloud computing, Hadoop, and Android. §3 gives the problem statement and establishes the motivation for a mobile-cloud computing platform. In §4, we list our assumptions, develop requirements, and justify using Hadoop for mobile-cloud computing. §5 describes the implementation of Hyrax, including our assumptions, requirements, and choices in configuring and customizing Hadoop for mobile devices. §6 describes our experimental evaluation of Hyrax. §7 describes our distributed multimedia search and sharing case study. §8 discusses the related work.

[1] The hyrax is a small herbivorous animal that lives in Africa and the Middle East. It is described as being the closest living relative to the elephant.

Chapter 2

Background

2.1 Smartphone technology

Advances in mobile hardware and software have allowed users to perform tasks that were once only possible on personal computers and specialized devices like digital cameras and GPS personal navigation systems. Using smartphones like the Apple iPhone, Android phones, and the BlackBerry, mobile users can now make full use of the Internet, capture and manage photos and videos, play music and movies, and play complex games. They have nearly ubiquitous access to the Internet via 3G services, WiFi, and peer-to-peer networking and can switch between networks automatically.

Sensors enable many interesting applications on smartphones. They provide information about the location, movement, and orientation of the phone and the environment's temperature and lighting. For example, the Google Android G1 contains an accelerometer, a GPS device, and a digital compass. Applications use local sensor data to customize and enhance the user's experience in a context-aware manner. For example, map applications display the user's current location on a map, rotate the map according to the user's heading, and provide customized directions. Games use motion data as input to create an immersive experience. Music applications such as Ocarina use microphone signals to simulate the effect of blowing into a musical instrument.

In addition to sensor data, smartphones are used to store, generate, and share multimedia data. Using built-in cameras and microphones, these devices can record photos, videos, and sound clips. Smartphones are also used to play movies and music downloaded to the device by the user. Several gigabytes of multimedia data can be stored locally on smartphones thanks to cost and size improvements in flash memory Matt.

Smartphones are becoming extremely widespread. According to International Telecommunication Union [2009], there are currently over 4 billion mobile subscriptions in the world. Among the devices used by these subscribers, smartphones are becoming increasingly common, accounting for a larger percentage of the mobile phone market and replacing less capable mobile phones. According to Hamblen, smartphones accounted for more than 14% of all mobile device shipped in 2008 and will account for 17% of all mobile devices in 2009. 116 million smartphones were shipped in 2007, 171 million were shipped in 2008, and a projected 203 million will be shipped in 2009. As powerful smartphones become more popular, it is increasingly feasible to run more complex software and more computationally-expensive tasks on them.

2.2 Cloud computing

Cloud computing is a style of computing in which dynamically scalable resources are provided as a virtualized service Knorr and Gruman. It allows service providers and other users to adjust their computing capacity depending on how much is needed at a given time or for a given task.

According to Myerson, cloud computing requires three components: thin clients, grid computing, and utility computing. Thin clients are applications that make use of the virtualized computing interface. Users are commonly exposed to cloud computing systems through web interfaces to use services such as web-based email, search engines, and online stores. Grid computing harnesses the resources of a network of machines so that they can be used as one infrastructure. Utility computing provides computing resources on demand, where users “pay as they use”. This is exemplified by Amazon EC2, which allows users to allocate virtual servers on demand, paying an hourly fee for each allocated server.

In mobile-cloud computing, the same type of virtualized interface is provided to users, but the system is ultimately supported by mobile devices or a combination of mobile and static devices. The possibility of heterogeneous clusters of servers and mobile devices in which the capabilities of each are used in conjunction is not excluded. The motivation for mobile-cloud computing is developed in §3.3.

2.3 MapReduce and Hadoop

MapReduce Dean and Ghemawat [2008] is a programming model and implementation developed by Google that is used to process very large datasets distributed across a cluster of servers. It is highly scalable, fault-tolerant, and useful for many large-scale data processing tasks. It is typically used in conjunction with the Google File System Howard et al. [2004], a distributed filesystem designed for “large distributed data-intensive applications.”

To use MapReduce, users specify a “map” function that takes an input key/value pair and outputs intermediate key/value pairs. For every key in the set of intermediate pairs, a set of values is collected. The user specifies a “reduce” function that processes each intermediate key/value set pair and generates an output. Input data is loaded from the distributed filesystem and output data is written back to the filesystem.

The MapReduce runtime system handles splitting the input data, scheduling map and reduce tasks, and transferring input and output data to the machines running the tasks. Jobs are managed by a master that assigns tasks to slave machines and provides the locations of intermediate values to reduce tasks. Computation on the machine where the input data is already stored is preferred in order to minimize network transfers. Large data transfers are performed directly between the machine where the data is stored and the machine that needs the data. Data transfers between machines on the same rack are preferred to transfers between machines that are more “distant” from each other in the network.

Hadoop Apache is an open source implementation of MapReduce used by many organizations for large-scale data processing. Hadoop is written in Java and operates on data stored in a distributed filesystem, usually the Hadoop Distributed Filesystem (HDFS), which is based on the Google File System. Hadoop instances consist of four types of processes Borthakur [2007]: NameNode, JobTracker, DataNode, and TaskTracker. There is one NameNode and one JobTracker in a Hadoop cluster. The NameNode maintains a directory of data blocks that make up the files in HDFS. The JobTracker manages jobs and coordinates sub-tasks among the TaskTrackers. Both a DataNode instance and a TaskTracker instance run on each worker machine. The DataNode stores and provides access to data blocks, and the TaskTracker executes tasks assigned to it by the JobTracker. Clients access files by first requesting block locations from the NameNode and then requesting blocks directly from these locations. The layout of these processes in a typical Hadoop cluster is summarized in Figure 2.1.

Hadoop is capable of tolerating faults by re-executing failed tasks (and tasks whose results are no longer available because of later failures) and by maintaining block replicas among several DataNodes. When speculative execution is enabled, the same task may be executed on multiple nodes to increase the probability of successful and fast results. Hadoop’s fault tolerance, along with its peer-to-peer bulk data transfers and largely independent tasks, allow it to scale to thousands of machines.

Hadoop is a cloud computing infrastructure in that it provides a virtualized interface to

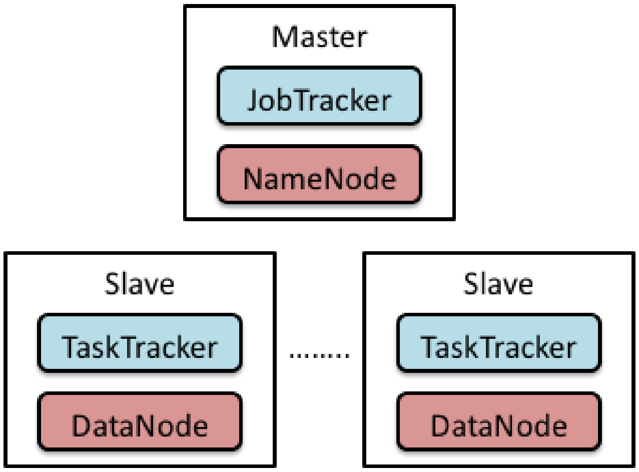


Figure 2.1: Typical Hadoop cluster configuration.

an arbitrarily scaled computing cluster. Hadoop programmers only need to be concerned with defining a high-level workflow for the system. The Hadoop runtime determines how to divide jobs submitted by the user into sub-tasks, where to physically store data, how to move computations and data, how to handle machine failures, and all of the other details that are required for a distributed computing system to work.

2.4 Android

Android Open Handset Alliance is an open source mobile operating system developed by Google and the Open Handset Alliance. It is built on top of the Linux kernel and provides an SDK for application development in Java.

Android uses the Dalvik Virtual Machine to execute applications. Dalvik is optimized to run on devices with constrained CPU, memory, and power resources. It implements a subset of Java 2 Platform Standard Edition (J2SE) using libraries from the Apache Harmony Apache Java implementation, giving it an advantage over other mobile platforms that only support Java 2 Platform Micro Edition (J2ME), which is limited by comparison. Java class files must be compiled to Dalvik bytecode (`.dex` format) and packaged in a `.apk` file in order to be used on Android.

Android provides an interface to system devices and services through a set of Java packages, including `android.os`, `android.hardware`, `android.location`, and android.media. This makes it easy to access and operate on multimedia data, sensor values, system resource usage data, and location information. Unlike some mobile operating systems, Android applications can use the filesystem directly, making it possible to manage files as on a traditional Unix system. Android also provides a shell interface, but it lacks many of the abilities of a typical Linux shell. Some of the missing utilities can be added by installing BusyBox Denys Vlasenko.

Chapter 3

Problem Statement and Motivation

3.1 Problem statement

We study the problem of how to create a mobile-cloud computing infrastructure that allows applications to utilize the collective data and computational resources of networked smartphones, particularly by modifying an existing non-mobile platform. The following questions are addressed:

1. In what ways does an existing cloud computing platform succeed and fail to meet the needs of a mobile deployment? Can it be modified to be adapted to be suitable, and how? The effectiveness of Hadoop on the Android platform is evaluated, and customizations of Hadoop for mobile hardware are implemented.

2. To what extent do mobile hardware and software reduce the effectiveness of an existing cloud computing platform? The performance of Hadoop on the Android platform is evaluated.

3. Does sharing processing and data among mobile phones in local networks reduce strain on globally-limited networks and decrease distribution latency by avoiding this bottleneck? The latency of resource usage is compared between sharing data through a central service and sharing it through a distributed filesystem.

4. What are the challenges in porting an existing cloud computing platform to run on smartphones? The obstacles that were faced in porting Hadoop to run on Android are reported.

5. How can a cloud interface to mobile resources be used effectively in practice? A distributed multimedia search and sharing application is implemented, and the advantages of using Hyrax instead of an ad hoc approach are reported.

With respect to this problem, we make the following claim:

Thesis statement. *It is feasible with today's mobile hardware and network infrastructure to provide mobile cloud computing using local data and computational resources to support larger system-wide goals.*

The thesis statement is validated through the design and implementation of a mobile-cloud computing system based on MapReduce, various performance experiments, and the development of a case-study application.

3.2 Goals and non-goals

In order to address the problem statement, our goals are the following:

- Motivate mobile-cloud computing by discussing the advantages of using mobile devices for cloud computing, proposing example applications, and showing that it is feasible using today's mobile technology.
- Implement Hyrax, a mobile-cloud computing platform, by porting Hadoop to run on Android smartphones.
- Evaluate what effect the mobile hardware platform has on the performance of Hadoop by developing a testbed and running a set of experiments on it.
- Determine whether Hyrax offers advantages in sharing data and processing compared to current approaches.
- Implement an application on top of Hyrax and evaluate it.

We do not aim to do the following:

- Implement a mobile-cloud computing platform that is fully optimized and ready for real-world deployment.
- Create a platform that is useful for generic distributed computation. We do not expect to compete with traditional server clusters for generic large-scale distributed data processing, only to support applications that make use of mobile-specific capabilities and data that is already on mobile phones.

3.3 Motivation

Mobile-cloud computing is motivated by the unique advantages of mobile devices, the wide range of applications that a mobile-cloud computing platform would facilitate, and the feasibility of such a platform using today's mobile technology.

3.3.1 Advantages of cloud computing on mobile devices

As defined in §1, mobile-cloud computing is cloud computing in which the foundational hardware consists at least partially of mobile devices. Traditional cloud computing systems are built on clusters of servers. Massive amounts of data are placed on these clusters through layers of virtualization, and then high-level jobs are executed to process this data and return useful results. In mobile-cloud computing, data originates and is processed on mobile devices.

Despite the obstacles that mobile computing systems inevitably face relative to stationary computing systems, including resource limitations, risk of loss and damage, variability in connectivity, and finite energy Satyanarayanan [1996b], there are numerous advantages of cloud computing on mobile hardware. These provide the core motivation for Hyrax:

- Mobile data such as sensor logs and multimedia data are immediately available and can be processed in-place or another node that is nearby in the network. Processing data in this way eliminates the need to expensively transfer data to remote, centralized services.

- Data can often be shared more quickly and/or less expensively among mobile devices through local-area or peer-to-peer networks. Data sharing is inherently useful in some applications, and it is needed for collaborative computing jobs. Distributing data using the local network avoids file uploads to and downloads from remote Internet services, which induce and are susceptible to global network contention.

- Services such as websites that use mobile data can be created with little extra computing infrastructure. Instead of hosting data and services on an expensive server farm or utility computing service, work can be distributed among mobile devices. With Hyrax, services would only need to act as a frontend to the mobile cloud.

- As stated in §2.1, billions of mobile devices are in use, and the proportion of these devices with smartphone capabilities is increasing. A mobile-cloud computing infrastructure could potentially be scaled to many more machines than a traditional cloud computing infrastructure simply because of the number of these devices that are in use.

- Ownership of the cluster hardware is distributed. By using mobile hardware owned by many different people, risks that arise when proprietary cloud services are used, such as data lock-out and dependence on external entities for data privacy, are avoided. Furthermore, maintenance of mobile devices in the cluster is also distributed since owners of smartphones almost always need them to be turned on and working properly. Note that distributed ownership also creates security and privacy challenges.

3.3.2 Applications

We are interested in applications that use data distributed among multiple phones such as multimedia files and sensor logs. Hyrax would support requests from these applications, either for direct access to the data or the results of running some job using the data. A highly appropriate application for Hyrax which incorporates both multimedia and sensor data is described in §1. In this section, more applications that Hyrax would facilitate are described.

Sensor data applications

Sensor data is composed of series of readings generated by a smartphone's sensors, such as the GPS device, accelerometer, light sensor, microphone, thermometer, clock, and compass. Each reading is timestamped, allowing it to be linked with readings from other sensors and multimedia files. Applications would use this sensor data by executing queries on the data as in a sensor database system Bonnet et al. [2001]. The data would be accessible via an interface similar to that of a relational database and large data transfers would be avoided by doing computations in-place (where the data is located) whenever possible. For example, a query might ask "what was the average temperature of nodes within five miles of my home at noon?" or "what is the distribution of velocities of all nodes within half a mile the next highway on my current route?".

The following applications would use sensor data in this way:

- **Traffic reporting.** This application would use location and movement data collected on mobile phones to infer traffic flow. The movement signal for a given time range would be processed (smoothed, interpolated) on the phone on which it resides, and a smaller result would be returned to the client. As with any traffic monitoring system, this application would be useful to drivers who need to navigate through traffic and officials in charge of controlling traffic. However, using mobile devices would allow for more precise monitoring than current systems provide. Traffic monitoring systems using sensor networks were implemented in Hull et al. [2006] using application-specific sensors and in Lo et al. [2008] using mobile device sensors.

- **Sensor maps.** This application would plot sensor levels such as temperature or sound levels on a map. The number of phones sampled would depend on the zoom level of the map. This could be used, for instance, to estimate levels of danger in a crisis situation in terms of temperatures, light, and noise levels and how they have been changing over time, or to visualize mobile usage distributions in a city as in Reades et al. [2007].
- **Network availability monitoring.** This application would collect network connectivity information on each phone by time and location. This could be used to determine where local-area and wide-area wireless network connectivity is available and how strong the signal of each is in a given location. This information would be useful for both wireless users and wireless providers. For example, Hull et al. [2006] analyzes WiFi availability using sensors attached to cars.

Multimedia applications

Multimedia data consists of files recorded on mobile devices, including videos, photos, and sound clips. It also encompasses files stored on mobile devices for entertainment, such as music and movies. Examples of applications that would use multimedia data are:

- **Similar multimedia search.** This application would find photos, videos, or music files whose contents are similar to that of an input sample. Each phone would reduce the dimensionality of its resident multimedia files locally using some given feature extraction algorithm, for instance using methods surveyed in Faloutsos [1996], and forward the result. Shazam Shazam Entertainment Ltd is a popular mobile application that does something similar, searching for songs similar to an uploaded music clip in a central database.
- **Event summarization.** This application would splice video clips from multiple devices into a single video which captures the entire event. The final video can be uploaded to the Internet or shared among mobile peers. For instance, this system would have been useful the protests that resulted from Iran's June 2009 presidential

election, where video clips were scattered in time and difficult to share as a result of the government's efforts to crack down on protests and the spread of information.

- **Social networking.** Sharing pictures has become a cornerstone of social networking websites such as Facebook Facebook. A mobile cloud could be integrated into the infrastructure of a social network to provide automatic sharing and peer-to-peer multimedia access while reducing the need for huge numbers of servers to store, back-up, and serve all of this data.

3.3.3 Feasibility

Mobile-cloud computing is enabled by recent advances in mobile hardware and software. CPU speed and RAM capacity have been approaching those of the desktop machines of less than a decade ago. For instance, the HTC Magic, a recently released Android phone, features a 528 MHz processor and 288 MB of RAM HTC [b]. Networking capabilities are also advancing. Many mobile devices can now connect to WiFi networks, which are widely available in homes and public areas. 3G networks provide widespread access to the Internet with speeds approaching that of WiFi. WiFi and 3G technologies are compared in depth in Lehr and McKnight [2003]. In addition to WiFi and 3G, Bluetooth allows low-power data transfer between devices with bandwidth that will soon approach those of WiFi networks, allowing fast, power-efficient bulk transfers between devices Bluetooth SIG [2009]. Smartphones will soon be able to create peer-to-peer networks using ad hoc WiFi and Bluetooth connections.

Thanks to increasingly powerful mobile hardware, mobile devices are now capable of running full-fledged operating systems such as Linux and Mac OS X. Mobile versions of these operating systems provide SDKs for writing complex applications using extensive libraries. The iPhone SDK supports development in Objective-C and C, allowing applications to make use of almost any existing code in these languages. Similarly, Android's Dalvik VM implements many J2SE classes, allowing existing Java libraries to be used in mobile applications with slight modifications. As a result, it is increasingly feasible to port desktop and server applications directly to mobile devices.

Unfortunately, energy density in batteries has not improved at nearly the same rate as computational capabilities in mobile devices Lehr and McKnight [2003]. This presents a serious obstacle for running services like a distributed computing platform on smartphones. As a result, power constraints are an extremely important consideration in designing a realistic mobile distributed computing platform.

Chapter 4

Approach

In this section, we explicitly state our assumptions, enumerate the requirements for a mobile-cloud computing platform, and explain why we chose to use Hadoop as a basis for Hyrax.

4.1 Assumptions

Our work depends on the following assumptions about the targeted hardware and how the system will be used. We briefly explain why we made each assumption. Future work may allow these assumptions to be relaxed.

- The system will be used primarily for computations that involve data on mobile devices, not for generic distributed computation. We do not expect to replace or effectively collaborate with traditional servers for generic large-scale computation. This is a reasonable assumption given that the fixed cost of computing resources is now very low using systems like Amazon EC2, which provide access to machines that are far more capable than smartphones.
- The smartphones under consideration have sufficient space to store multimedia data and sensor logs, on the order of several gigabytes. This is not an unreasonable assumption given the availability of cheap flash memory Matt. In our testbed, for instance, an 8 GB microSD card costing about $15 is installed in each phone.
- A central machine that can connect to each phone exists. This is required to use Hadoop without extensive modification since Hadoop's NameNode and JobTracker

processes must each run on some machine. This assumption is not unrealistic given how widespread Internet connections on smartphones are. Note that these do not have to be high-bandwidth connections since the NameNode and JobTracker are only used for coordinating data and jobs.

- Files shared on the mobile network will not be modified often if at all. Our system is targeted at multimedia and sensor data, which can be considered historical records that do not need to be changed.
- Each smartphone is reachable from each other device in the network via IP. This is an unrealistic assumption given the complexities imposed by firewalls and network address translation (NAT). For instance, when a mobile device is behind a wireless router, it is only possible for other devices to connect to it (using plain TCP or UDP sockets) if the correct ports on the router are forwarded to the device. However, several peer-to-peer NAT- and firewall-traversing protocols exist, including the Session Initiation Protocol (SIP) Rosenberg et al. [2001], JXTA Sun Microsystems [a], and SmartSockets Palmer et al. [2009]. These protocols use mutually-accessible proxies to coordinate transfers on nodes behind firewall and NAT layers. This problem is not addressed in our implementation, but it would not be difficult to incorporate an existing peer-to-peer protocol into Hadoop, especially one such as JXTA that provides a Java SocketFactory implementation.

Note that the following are not assumed:

- Static network topology. It is not assumed that devices in the network, other than the central server, will be present throughout a job.
- Homogeneous hardware. Heterogeneous clusters of devices, including both mobile devices and traditional servers, are allowed.

4.2 Requirements

A mobile-cloud computing platform must satisfy the needs of the applications written for it while using resources efficiently. The essential functionality of a mobile-cloud computing system is:

- **Global data access**. Applications should be able to access any data that the user of the application has permission to access regardless of the physical nature of the data, for instance where it is stored and how it is replicated.

- **Distributed data processing**. Given a program that takes data on the filesystem as an input, the platform should be able to compute the result of executing this function on the appropriate data and make the results available to the requester.

In order for the system to usefully provide global data access and distributed data processing in a real-world mobile distributed system, it must also have the following features:

- **Fault-tolerance**. It is important for the system to tolerate mobile devices leaving and entering the network. Individual devices are susceptible to network signal loss, running out of battery power, being too far away from other phones for peer-to-peer networking, and hardware failure.
- **Scalability**. The system must scale with an increasing number of devices and an increasing amount of data. The latency of an operation invoked on the system should increase at most linearly with respect to the amount of the data being processed or accessed. Increasing the number of phones should have a positive to neutral effect on job latencies.
- **Privacy**. File owners should be able to control other users' access to their data. For instance, users should be able to specify which other users have access to individual pictures taken on their phones.
- **Hardware interoperability**. Machines that the software components of the system runs on should be able to interoperate with other machines regardless of hardware specifics. Different types of mobile devices and servers should be able to work together as long as they run compatible versions of the software.

The implementation of this system should use mobile resources wisely, including:

- **Battery life**. Energy is a finite resource on any mobile device. A service that runs for a long period of time on a mobile device must be especially conscious of energy usage. Energy density in batteries has been increasing much more slowly than the capacities of other mobile computing resources, so preserving energy will likely be the chief priority for mobile software systems for a long time Estrin et al. [2002].
- **Network bandwidth**. Wireless network connections on mobile phones are relatively slow and intermittent, and they account for a significant percentage of power consumption. In fact, network transmission is orders of magnitude more energy-costly than CPU cycles Palmer et al. [2009]. Therefore it is often more efficient to process data on the phone where it resides and return a smaller result.

Furthermore, on mobile data networks, bandwidth is a globally limited resource. The more data that devices send, the slower and less available the service becomes for everyone. As stated in §1, this can make it difficult to transmit data when an extremely large number of mobile devices are being used in the same location.

- **CPU cycles and memory**. Using the processor on a mobile device requires energy from the battery and may interfere with the performance of other applications. Along the same lines as processor usage, excessive memory usage may interfere with the performance of other applications. Memory allocation tends to be more tightly constrained on mobile operating systems than it is on traditional operating system configurations. For instance, Android limits the heap size of each application to 16 MB.

  In the case of Java-based mobile application frameworks such as the Android SDK, it is especially important to avoid CPU-intensive operations. The virtual machine (the Dalvik VM in the case of Android) adds an extra layer of abstraction which greatly impedes the performance of a program that is CPU- and/or memory-bound compared to the equivalent program running directly on the hardware.

- **Time**. Time-efficiency is always important to users. The platform should be able to compute results in a reasonable amount of time. This is particularly important in mobile applications because many mobile operating systems only allow the user to focus on one application at a time.

- **Storage**. The size and cost of flash storage is improving, but the amount of permanent storage available on mobile devices is still limited compared to the amount of storage on traditional machines. Furthermore, “erase” and “write” operations on flash memory cause memory wear Corsair, reducing data integrity over time. Therefore permanent storage should be used conservatively.

Above basic resource considerations, there are several constraints inherent in mobile computing that need to be considered, especially when transforming a distributed system to one that is both distributed and mobile, as we do with Hadoop. These challenges, originally outlined in Satyanarayanan [1996b], are:

1. Mobile elements are resource-poor relative to static elements. The additional weight, power, and size restrictions compared to static counterparts will always have a negative effect on performance and capacity.

2. Mobility is inherently hazardous. Mobile devices are more susceptible to loss and damage.

3. Mobile connectivity is highly variable in performance and reliability. Wireless networks vary in speed and reliability, and mobile users constantly move between networks. There is often no network available.
4. Mobile elements rely on a finite energy source. Battery power consumption must be considered at all levels for conservation to be effective.

We must demonstrate how Hyrax satisfies or fails to satisfy (in its current state) each of these requirements.

4.3 Using Hadoop for mobile-cloud computing

In order to satisfy the requirements that have been outlined, a new infrastructure had to be built from scratch, or an existing one had to be modified. Recognizing that Hadoop implements the core required functionality, we decided to use it as a starting point. Hadoop has several advantages and disadvantages with respect to the requirements of mobile-cloud computing.

4.3.1 Advantages

Hadoop implements much of the core required functionality outlined in §4.2, including global data access, distributed data processing, scalability, fault-tolerance, and data-local computation (and thus efficient use of network resources).

HDFS, as a distributed filesystem, provides global data access to all devices in the network. Furthermore, data blocks are transferred point-to-point, not through an intermediary. As a result, the speed of data transfer is limited primarily by the network bandwidth between the two devices involved.

Distributed data processing is provided via Hadoop's MapReduce implementation, which divides jobs submitted by the user into independent "tasks" and distributes these tasks to slave nodes, taking the physical location of input data into consideration. These slave nodes execute map tasks on data stored on HDFS. As the outputs of map tasks become available, reduce tasks process this intermediate data and write results back to HDFS. When possible, data that is physically located on a given node is processed on that node, avoiding data transfers.

Hadoop was designed to scale to thousands of machines, and has been shown to do so by Yahoo! Yahoo!. It is also designed to tolerate faults; any sufficiently large system faces hardware failures with some expected frequency. HDFS implements file permissions, which can be used to protect user data from unauthorized access. Therefore Hadoop covers our requirements for scalability, fault-tolerance, and, to some extent, privacy. Furthermore, the hardware fault-tolerance of Hadoop covers the mobile challenge, "mobility is inherently hazardous".

Since Hadoop uses abstract IPC interfaces for communication between processes and between physical nodes, it is trivial for different types of machines running Hadoop processes to work together. Therefore Hadoop also satisfies the "hardware interoperability" requirement.

Although Hadoop does not currently take energy efficiency into account, there are signs that this will change. Many companies are interested in data center efficiency Google, Intel for reducing costs and the environmental impact of their operations. Therefore energy usage improvements in Hadoop may be implemented in the future. In fact, researchers have begun to investigate and suggest improvements for energy efficiency in Hadoop. Chen et al. [2009] offers some suggestions for increasing the energy efficiency of Hadoop, such as increasing the replication factor, avoiding excessive fan out and fan in, and making sure that the intermediate pairs processed by reduce workers fit in memory. Given Hadoop's popularity among major companies which are constantly optimizing their processes and the research that is being done to improve Hadoop's power efficiency, it is likely that Hadoop will become more power-aware and power-efficient in the future. Furthermore, investigating the battery consumption of Hadoop on mobile devices may yield insights useful to improving the energy efficiency of Hadoop in a traditional setting.

4.3.2 Disadvantages

Despite numerous advantages, Hadoop is less than ideal for some of the mobile aspects of mobile-cloud computing. It implements much of the functionality that our platform requires, but it does not cover all of the requirements. This is mostly because Hadoop was designed and implemented with commodity server hardware in mind rather than resource-constrained hardware.

One problem is that Hadoop is not conservative in CPU and memory usage. Hadoop was designed for I/O bound jobs, i.e. those in which reading, writing, and transferring data are the most time-consuming operations. Hadoop's liberal use of CPU and memory is exemplified by several aspects of its codebase. For example, Hadoop makes heavy use of interfaces and inheritance, which impose computational overhead because of the lookups that are required to determine which function to execute. Android provides several guidelines for writing efficient code, such as avoiding object instantiation, avoiding internal getters and setters, and preferring "virtual" over "interface" Android [a]. Of course, Hadoop was not written with these guidelines in mind since it was developed for normal JVMs running on traditional hardware. By default, Hadoop assumes that memory buffers on the order of 100MB can be allocated, such as in map output buffering. This is clearly not the case on mobile devices.

In the interest of avoiding "reinventing the wheel", Hadoop also uses technologies that are not well-suited for mobile devices. For instance, it uses XML extensively, which is notoriously expensive to parse. It also uses servlets to serve intermediate results, even though a light-weight custom HTTP server would require less overhead. JSPs, which require dynamic compilation, are used to provide a monitoring interface to DataNodes and TaskTrackers. This inefficiency is magnified on a mobile device.

Hadoop is also lacking in its ability to cope with varying and slow network conditions. Hadoop is typically run on servers Apache connected via 1 Gbit/s to 10 Gbit/s Ethernet networks, which are about 8 and 80 times faster than 802.11g WiFi respectively IEEE [2003] and much more stable. A network-bound MapReduce job may take approximately this many times longer to complete using WiFi connections and perform even worse when signal is poor for some of the nodes. This network bottleneck would occur during the shuffle phase of MapReduce, where intermediate key-value pairs are distributed among the nodes. As a result, for a typical MapReduce job, with all other things equal, a wireless mobile cluster would be expected to perform much worse than a traditional cluster. MapReduce jobs that run on mobile device networks would have to be tailored to low-bandwidth conditions, e.g. by making sure that intermediate keys and values are small.

The design of HDFS precludes disconnected operation, a major feature in other mobile distributed filesystems Kistler and Satyanarayanan [1992]. DataNodes store blocks without any knowledge of the file paths that they correspond to. Therefore it is impossible for mobile devices to access data on HDFS, even if it is stored locally, if no connection to the NameNode exists. Mobile applications would have to use a separate filesystem, most likely the local native filesystem, as a backup when the NameNode is not reachable. Similarly, mobile applications cannot execute compute jobs through the MapReduce interface when disconnected from the JobTracker. A side-effect of this limitation is that Hyrax does not face consistency problems such as how to resolve changes made to the same file during disconnected operation.

4.4 Hadoop's assumptions in relation to mobile computing

It is important to consider the assumptions of the Hadoop and HDFS architecture Borthakur [2007] in relation to mobile devices and the requirements of our platform. These assumptions should be reasonably compatible with the particulars of smartphones. They are:

- **Hardware failure is common.** In a mobile device network, failure is equivalent to a device being disconnected from the network for an extended period of time, which is a common occurrence. For instance, mobile devices sometimes disconnect from WiFi when they are in an idle state in order to save power Android [b]. Mobile devices also become disconnected when they enter places with poor wireless signal coverage, such as basements, rural areas, and airplanes.

- **Applications use large datasets.** By default, HDFS uses 64 MB blocks to store files. This is much larger than most multimedia files collected on phones with the exception of large video files. For example, a photo taken using the Android G1 phone uses about 1 MB, and a typical MP3 music file uses about 3 MB. Since at least one block must be allocated per file, storing individual photos and songs as files on HDFS may waste a lot of space and lead to excessive block lookup requests to the NameNode. This overhead can be reduced by combining multiple multimedia files into larger files to be stored on HDFS.

- **Applications do not require low-latency access to results.** HDFS is designed to support batch-processing rather than interactive use. This assumption might seem difficult to reconcile with the demands of mobile applications, where applications might execute custom queries and expect a result within a short time. However, there are many cases where mobile applications could, instead of executing Hadoop jobs directly, send queries to an intermediary which periodically runs a small set of common queries and caches the results.

- **Files are not modified after they are created.** This fits well with multimedia files and sensor logs, which are generally not modified once they have been captured. They are essentially historical records.

- **Moving computation is easier than moving data.** This assumption fits extremely well in a mobile environment. As noted in §4.2, bandwidth is a precious resource on mobile devices, and code generally takes up less space than the data that it is used to process. Therefore it often makes sense to distribute instructions for each phone to execute on the data that it already contains instead of having each phone offload all of its data to remote machines.

In summary, the assumptions of hardware failure, file non-modification, and relative efficiency of moving code instead of data fit very well with a mobile environment. Hadoop's job latency assumptions require applications to be developed such that MapReduce jobs are executed periodically instead of on-demand, which should be acceptible in most cases. Hadoop's dataset size assumptions do not fit very well and thus need to be accounted for.

4.5 Using Android for Hadoop

Google's Android operating system was the most natural choice of a mobile platform to run Hadoop on. Android's Dalvik VM implements a subset of the Apache Harmony Java implementation, which includes most of the Java classes used by Hadoop. Hadoop, an Apache project itself, depends on several Apache libraries, such as Apache log4j, Apache XML, and Apache Commons. Because of this compatibility, it was possible to port Hadoop without rewriting a huge amount of code.

Note that most mobile devices that run Java only implement J2ME, which is not sufficient for running an application like Hadoop without extensive modification. J2ME does not include many of the high-level networking, process management, and file I/O features that Hadoop depends on. Using Android made it possible to port Hadoop without completely overhauling its source code.

Another appealing aspect of Android is its open nature relative to other mobile platforms such as the iPhone, which is the most popular mobile application platform. Android allows arbitrary applications to be installed on any number devices without any external permission. In contrast, the iPhone SDK requires an expensive developer account in order to install an application on an actual iPhone. Android's debugging tool, ADB, can be used to execute shell commands, install applications, display phone logs, and push and pull files. Because it is a shell utility, it can be used in scripts. The iPhone provides application installation and log viewing within the XCode IDE, and it does not support the execution of arbitary shell commands or the creation of files. Overall, Android's relative open-ness makes it an acceptible platform for distributed mobile applications, whereas closed platforms such as the iPhone make the development of such systems more difficult.

4.6 Evolution of our approach

In the early stages of this project, we attempted to port Hadoop to SunSPOTs Sun Microsystems [b], which implements a J2ME API. We found this platform to be too limiting for a direct port of Hadoop; it is missing many of the basic J2SE-style Java classes that Hadoop depends on. For instance, the SunSPOT API does not even include the `List` interface. It would probably be easier to write a simple MapReduce platform specifically for J2ME devices from scratch than to port Hadoop. We chose not to implement a MapReduce system from scratch because we wanted to study a full-featured, real-world MapReduce implementation.

We found it much easier to make progress in porting Hadoop when we tried to do so using the Android SDK and Android devices. We were not certain that it would be possible to port every detail of Hadoop. In fact, we failed to port dynamic class loading and JSP serving. However, we were able to get the core functionality of Hadoop to work through a lot of painstaking debugging.

Chapter 5

Implementation

In this section, we discuss how Hadoop was ported and configured for Android smartphones and describe the obstacles that were faced. In general, we found that the challenges induced by Android resulted from a lack of openness and deviation from a the typical Unix interface whereas those induced by Hadoop resulted from assumptions about system performance. We found that there were many mobile-specific customizations that were made trivial by features of Hadoop.

5.1 Porting Hadoop

The first step towards porting Hadoop to run on Android was to compile Hadoop's source code into an Android application. We wanted to create an Android application that would act as a slave in a Hadoop network, running DataNode and TaskTracker instances. Instantiating the DataNode and TaskTracker was just a matter of including Hadoop's source in the Java build path in an Android project. We started with Hadoop 0.19.0. Of course, the system did not simply work immediately. Fixing the incompatibilities between Hadoop and Android was a very difficult debugging task because of several obstacles imposed by both Android and by Hadoop. These obstacles are described in §5.1.2 and §5.1.1. Having fixed most of these incompatibilities, the source was later patched from Hadoop version 0.19.0 to version 0.19.1. This did not break any of our modifications.

Note that the NameNode and JobTracker processes were not ported to run on Android. In Hyrax, these must be run on a traditional machine. However, it would not be any more difficult to port these than it was to port the DataNode and TaskTracker.

5.1.1 Android obstacles

Porting Hadoop to run on Android was a very difficult debugging challenge. Android imposes several additional constraints on its applications that are not present in a typical Linux system. Furthermore, since it uses a custom Java implementation that is not fully compatible with Sun's JVM implementation, legitimate error-free Java class files are sometimes rejected at runtime. As a result, Hadoop's source and that of many of its libraries needed to be changed, either by removing or rewriting offending code, for it to run as an Android application.

At runtime, Dalvik performs an additional consistency check that causes it to reject some opcode sequences even though they were accepted by the Dalvik bytecode compiler. This caused many classes packaged in libraries that Hadoop depends on to be rejected. To get around this, the source code for the incompatible libraries had to be downloaded, parts of the code that caused classes to be rejected had to be tracked down and removed without modifying important behavior, and the libraries needed to be recompiled.

The usual `java` executable is not available on Android. In Android, Java classes cannot be installed and executed directly from the command line. An application must always exist as an `.apk` file. Hadoop's launch script runs, on each worker node, DataNode and TaskTracker instances as separate processes. This was replaced with code that instantiates DataNode and TaskTracker objects within different service processes under a single Android application.

A related issue is that it is not possible to execute new JVM instances within an Android application using the shell. Hadoop does this to launch child worker processes. In Hadoop, the call to `java` on the shell was replaced with a call to the child process class's `main` method, passing in the appropriate arguments. Some of the code that manages the JVM instances spawned by Hadoop was also removed.

Android uses incompatible versions of the UNIX shell utilities needed by Hadoop. Hadoop makes calls to `df`, `du`, and `chmod` in managing files and reporting the amount of space available in a DataNode. In some cases, Android's versions of these utilities accept different input flags and produce outputs in a format that is different from the one expected by Hadoop. To work around this, calls to shell utilities were removed and replaced with calls to equivalent Java methods exposed by the Android SDK.

In order to execute arbitrary jobs on TaskTrackers, Hadoop must package and send Java classes to the TaskTrackers at runtime. When a TaskTracker executes a map or reduce job, it makes a call to `java`, adding to the classpath the path of the job's unpackaged classes. Since classes need to be converted to `.dex` format before they can be used in an Android application, classes cannot be easily loaded at runtime in this way. At this point, all of the the job class files are simply packaged into the Hyrax worker application. Dynamic class loading in Android may be implemented in the future, but it is not essential to addressing our research questions.

Finally, Android's debugging system caused Hadoop to run extremely slowly. In most cases, it was not fast enough to be useful since it would take too long to get to the point in the program execution where the bug occurred. This made debugging very slow and difficult in many cases.

It is important to note that not all of these issues arose on the Android emulators; some only occurred on actual phones. There are several important differences between actual Android devices and Android emulators. One is that the Android emulator gives root privileges to the user, whereas a typical (non-development) Android phone does not. Another is that the timing of various system operations is very different. Developing and running Hyrax on actual hardware exposed more problems than if Hyrax had only been tested on Android emulators.

5.1.2 Hadoop obstacles

There were also several assumptions made by Hadoop that caused faults and performance problems when it was run on Android.

Hadoop allocates memory buffers that are on the order of 10 to 100 MB. This is too much for an Android application, whose heap can grow to a maximum of 16MB. To fix this, these buffer sizes were reduced to about 1 MB. This caused excessive swapping to occur. This swapping was reduced by adjusting the `io.sort.record.percent` parameter (described in §5.4).

The default values for some timeouts in Hadoop are not long enough for a mobile device network. For instance, the value of the `dfs.socket.timeout` had to be increased to compensate for connection issues.

Hadoop uses XML for its configuration files even though the same key-value configuration could be stored in a simpler format, such as a properties file. XML parsing is generally expensive; this is even more apparant on a CPU- and memory-constrained smartphone. In fact, parsing the XML configuration files was the bottleneck in initializing the DataNode and TaskTracker instances. XML configuration files were replaced with properties files to speed up Hyrax.

5.2 Hadoop on a mobile cluster

Having ported the DataNode and TaskTracker processes to work on Android, a Hadoop cluster was configured to run on Android phones. Running Hadoop on a cluster of phones is analogous to running Hadoop on a cluster of servers. In both cases, there is one instance of the NameNode and one instance of the JobTracker. These often run on the same machine. The slave machines in the cluster each run DataNode and TaskTracker instances.

In Hyrax, the DataNode and the TaskTracker are run on each phone in separate Android "service" processes within the same application. Android applications may consist of multiple processes, some of which run as background services. Since the DataNode and TaskTracker are run as Android services, they can run in the background of other applications. The configuration of Hyrax is illustrated in Figure 5.1.

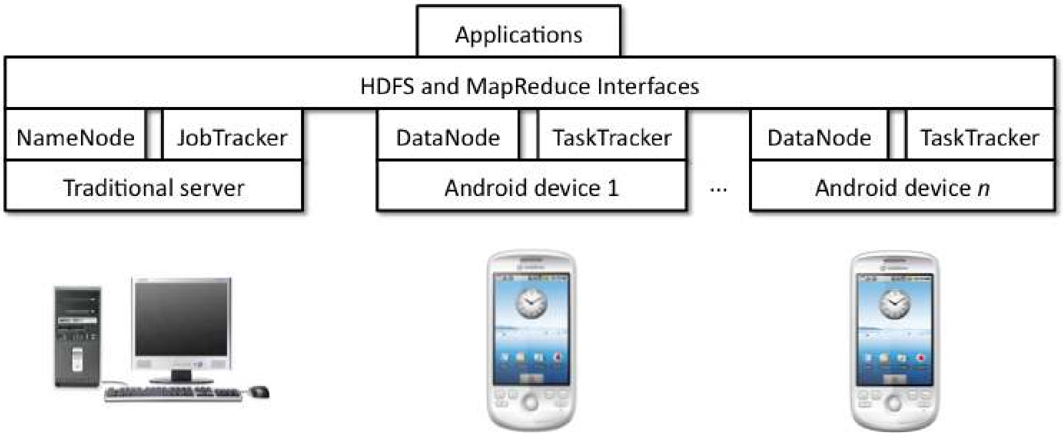


Figure 5.1: Hyrax hardware and software layers.

5.3 Mobile-specific components

In addition to the DataNode and TaskTracker services, threads that put each phone's multimedia data on HDFS and store sensor logs as files are spawned. In future work, Hive Apache [b] will be used to store sensor data in a more structured way, but coding obstacles have prevented Hive from being used at this point. As discussed in §3.3, it would be useful to be able to process sensor data as if it exists in a relational database. Hive provides a data warehouse infrastructure on top of Hadoop, providing a SQL-like query interface to the data and using MapReduce jobs to execute queries. This will be useful for storing and accessing sensor data.

For the purposes of our experiments, a thread which records (to the local filesystem) system load data, including power level and CPU, memory, network, and disk I/O statistics, is also spawned. Within the application, a server is run allowing external scripts to control data uploading, kill the program, and check the program status. Figure 5.2 illustrates the data interaction among all of the software components that run on each phone.

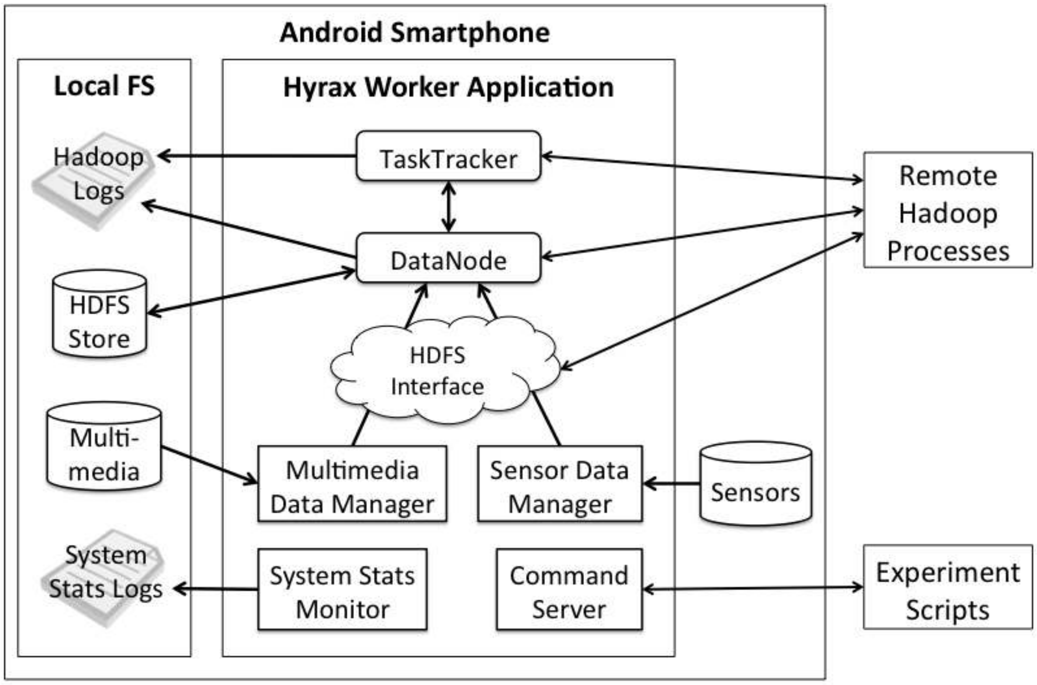


Figure 5.2: Hyrax worker application component interaction diagram.

5.4 Adjusting Hadoop's configuration parameters

Many of Hadoop's parameters were adjusted to suit a smartphone cluster. Through experience, these settings have been found to be appropriate for our mobile devices. In future work, experiments may be performed in which important parameters are varied independently to find the optimal setting for each one.

The amount of memory available for sorting map output key/value pairs (`io.sort.mb`) is greatly reduced because Android limits the memory available to a given application to just 16 MB. With everything else running in this application, it was not possible to allocate more than 1 MB for the map output buffer, which is much smaller than Hadoop's default of 100 MB. In the `MapOutputBuffer` class, which is used to collect map outputs, when this memory buffer is exceeded by the map outputs (which happens almost immediately when the buffer is small, at least for jobs that have many intermediate pairs), key/value pairs are swapped out to files on disk. On a server, this swapping is still relatively inexpensive because of the hard disk cache, which is 16 MB in each disk on our servers. However, on a mobile device using flash memory, there is no such cache. Reads and writes are placed in a small buffer and then flushed to the flash memory. Therefore swapping map outputs to disk is extremely expensive in Hyrax. In our experience, it causes jobs to execute about 100 times slower. Through benchmarks and profiling, it was determined that setting another parameter, `io.sort.record.percent`, which determines the percentage of storage used for records instead of key/value pairs, to 0.5 instead of the default of 0.05 reduced spilling significantly.

DFS block size (`dfs.block.size`) is decreased from 64 MB to 8 MB. The default of 64 MB is derived from the design of GFS Howard et al. [2004]. GFS uses such a large "chunk size" in order to minimize interaction between the client and the metadata server and allow for more total data to be addressed – the larger the chunksize, the fewer requests the client needs to make to the directory to find all the chunks for a given file, and the less metadata is needed per byte. Using a large chunk size also decreases network and metadata overhead. A drawback of a using large chunk size is that "hot-spots" may develop for the chunks of files requested by many clients.

Because of the networking and processing limitations of mobile devices, the DFS block size is reduced. It takes significantly longer to transfer a 64 MB block on a wireless network than on the wired networks typically used by Hadoop. Furthermore, the data files that are used on smartphones are generally much smaller than 64 MB, in which case the extra block space is unnecessary.

DFS DataNode socket write timeout (`dfs.datanode.socket.write.timeout`), DFS socket timeout (`dfs.socket.timeout`), and MapReduce task timeout (`mapred.task.timeout`) are increased to very large values in order to compensate for the additional time required for slower network transfers and CPU speeds.

5.5 Replication strategy

The replication factor r controls a tradeoff between battery consumption and data availability. The higher the value of r, the more network transfers need to occur to create $r - 1$ replicas on other devices in addition to a local replica. However, replication improves block access times during concurrent requests while spreading the load among many devices. Whether r saves or wastes battery and time is determined by how often individual files are accessed.

For every file f, a file-specific replication factor r_f is assigned. For each multimedia file f, $r_f = 1$ is used, meaning that the blocks of the file will only be stored on the original device, unless there is no space left on the device. A higher r would entail uploading the entire file to another device by default, which is expensive. For each sensor log f, $r_f = 1$ for the same reason. Sensor logs can grow to be larger than multimedia files over time, so sending them over the network should be avoided. Of course, the default r can be increased if saving power is less important than data reliability for some application.

On the other hand, using a low r puts a high load on devices that store popular blocks and puts these blocks at the risk of being lost if the device hosting them leaves the network. To avoid this, applications that use HDFS should adjust r_f depending on the popularity p_f of f. When p_f is low, r_f should remain low to avoid unnecessarily transferring the blocks of f. If p_f is high, then it makes sense to increase r_f to spread the load of serving f among more devices, increasing throughput and decreasing the average energy consumed on each device that hosts the blocks of f. Increasing r_f also decreases the probability that f will be lost, which should be avoided if p_f is high.

Individual users might want to specify r_f in order to, on one hand, preserve battery energy or, on the other, to make sure that some essential piece of data becomes widely available. For instance, in a crisis, combat, or protest situation, a single video or photo might be extremely important and at immediate risk of loss. The HDFS interface makes it easy to set r_f, and this setting could be exposed to mobile application users.

5.6 File organization

Files originating on mobile devices are organized in a straightforward way. Videos are placed under a “videos” directory, photos are placed under a “photos” directory, and each sensor log type is placed under its own directory.

Additional files containing information about each multimedia file, including the device that it originated from, its start and end time, and its type, are placed on HDFS. Start time, end time, and device information can be used to associate the file with sensor readings. In the future, this information will be stored using a Hadoop database system such as Hive Apache [b] or HBase Apache [a]. Using a database will facilitate efficient sensor value aggregation queries over time ranges. This would be useful, for instance, in determining properties like noise and velocity over the course of a video.

5.7 Heterogeneous networks

Because of the limited resource capacities and speeds of phones, it can be beneficial to add servers to augment the performance and reliability of the mobile cloud. Servers can be used to store replicas of mobile data, expedite MapReduce jobs, and serve data more quickly and without wasting battery energy. Because Hyrax exposes the same interfaces as a Hadoop DataNode and TaskTracker pair, it can inter-operate with servers in addition to other mobile devices. Therefore it is trivial to run hybrid clusters consisting of both mobile devices and servers.

We envision location- and event- specific applications in which wireless networks with locally connected servers are set-up to support mobile-cloud compute jobs and file sharing. For instance, at a sporting event, the stadium could provide a wireless network with attached servers, allowing broadcasters and to efficiently access and search through fans' videos and photos taken during the game. This would create a more interactive experience for fans. A network with the hardware that would be required for this has been deployed for Media [2009] in Pittsburgh's Mellon Arena to serve video playbacks to mobile devices during hockey games.

5.7.1 Server-augmented Block Replication and Serving

Since mobile devices and their network connections are slower and less stable than traditional servers and their network connections, it makes sense to replicate data to and serve data from servers when they are available to save time and resources. If a phone in the mobile cloud dies, its important files should remain available in the cloud. Furthermore, access to this data should be optimized for speed and mobile resource efficiency.

Using Hadoop's rack-awareness feature, it is easy to make sure that data is replicated to and served from servers whenever possible. In order to make sure that all data is replicated to servers, phone hostnames are mapped to "`/phone-rack`" and server hostnames (all non-phone hostnames) are mapped to to "/server-rack". By default, Hadoop's replication strategy when $r = 3$ is to store one replica on a node in the local cluster, one on a different node on the local cluster, and one on a node in a random remote cluster Borthakur [2007]. In the case of a hybrid phone-server cluster with 2 racks, this implies that the blocks making up any f for which $r_f \geq 3$ will be replicated to a server. By considering all clients to be on the server rack, blocks available on servers will be served to clients from servers instead of other phones (except when a block is already located on the client phone). Figure 5.3 illustrates a possible distribution of block replicas using this configuration.

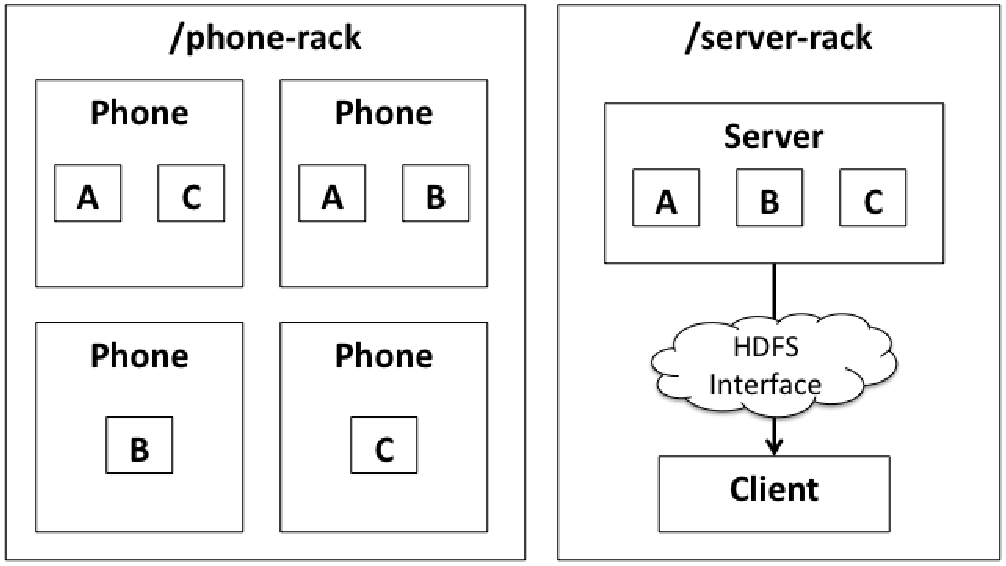


Figure 5.3: Example of block replica distribution in Hyrax with replication factor 3 for each file using /phone-rack and /server-rack.

Note that when servers are available in the cluster, there is no reason to replicate blocks to other phones in the cluster. This would happen under the default replication strategy using the /phone-rack / /server-rack configuration, which simply takes advantage of the default strategy. Instead, it would be better to put each phone on a separate rack (since transfers between phones are actually more expensive than transfers from phones to servers, shown in §6.3) and use a replication strategy that always places replicas on /server-rack, only placing replicas on other phones in this case.

In clusters consisting of phones distributed among multiple local networks, phones should be mapped to different racks depending on their network distance from other phones and servers. In this case, the directory structure of racks can be applied. For each local network of phones L, a rack `/rack_L` can be created. Each phone p connected to L would be assigned to rack `/rack_L/phone_p`, and each server s connected to L would be assigned to rack `/rack_L/servers`. This would encourage transfers between phones on the local network when transfers to servers on the local network are not possible. This scheme may be implemented in future work (see §9.1.5).

Chapter 6

Evaluation

In this chapter, Hyrax is evaluated in terms of how effectively it meets the requirements that were established in §4.2. The experiments are used to evaluate specific, quantifiable aspects of Hyrax, such as how it uses resources, tolerates faults, and scales. Hyrax is evaluated more qualitatively in §7.

Five experiments were performed. The first experiment compares the baseline performance of the smartphones and servers in our testbed. The second experiment compares the performance of Hyrax on smartphones and Hadoop on servers for MapReduce benchmarks. The third experiment determines the extend to which Hyrax tolerates devices leaving the network. The fourth experiment compares the performance of file sharing using Hadoop to file sharing through a remote service. The final experiment compares the battery usage of Hyrax to that of other applications.

6.1 Experimental infrastructure

6.1.1 Testbed

The testbed for conducting our experiments is a cluster that consists of 10 Android G1 (HTC Dream) phones and 5 HTC Magic phones, each running the Android 1.5 "Cupcake" platform. Three of the HTC Magic phones were faulty and were thus excluded from experiments. The Android G1 is equipped with a 528 MHz Qualcomm MSM7201A processor, 192 MB of RAM, a 1150 mAh lithium-ion battery, IEEE 802.11b/g connectivity, GPS, an accelerometer, and a digital compass HTC [a]. The hardware capabilities of the HTC Magic are similar to those of the G1. It includes a 528 MHz Qualcomm MSM7200A processor, 288 MB of RAM, a 1340mAh battery, and the same sensors and wireless capabilities as the G1 HTC [b]. An 8 GB microSD card is installed in each phone to store HDFS data, multimedia data, sensor data, system resource usage logs, and Hadoop logs. Since Android does not support peer-to-peer networking yet, the phones communicate with each other on an isolated 802.11g network via a Linksys WRT54G wireless router with no firmware modifications. The NameNode and JobTracker processes run on a desktop machine that is connected behind this router via Ethernet. The phones are connected via USB to a controller machine which executes experiment scripts. These scripts are used to install Hyrax, initialize the cluster, run benchmarks, and collect and post-process data.

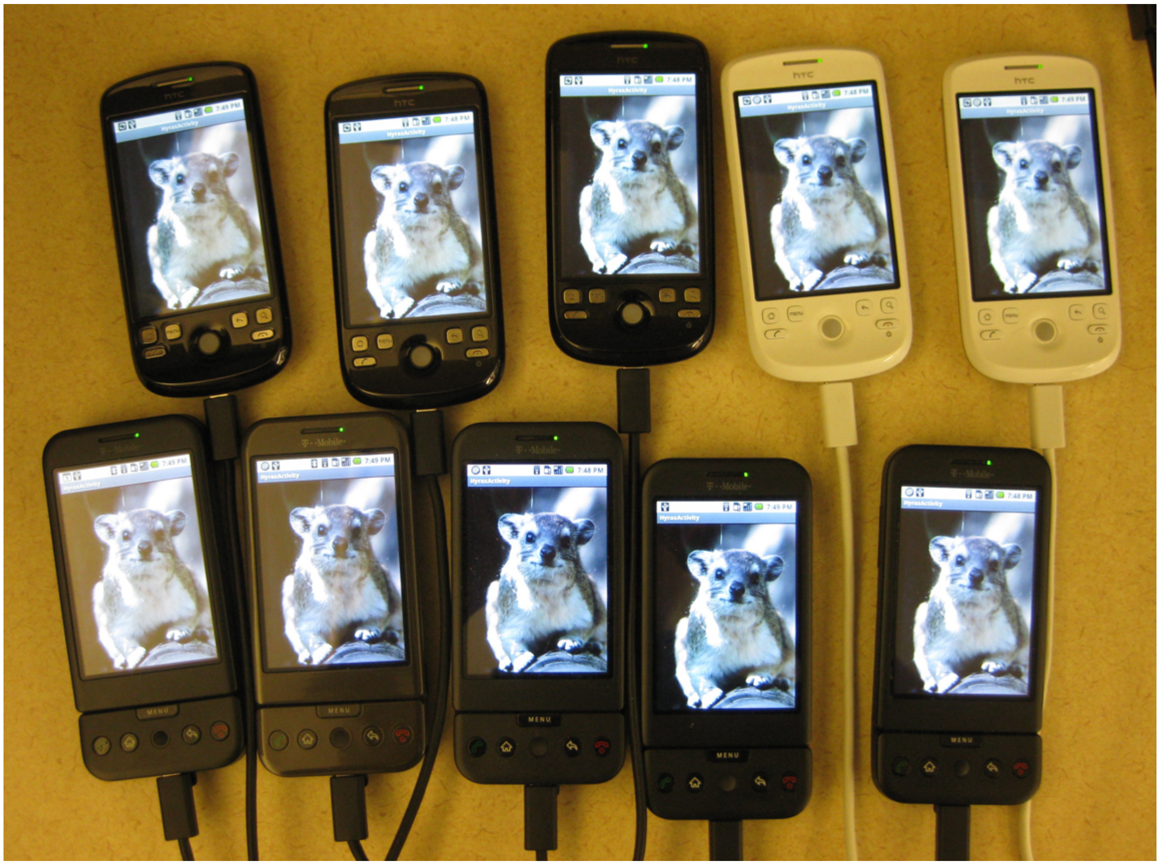


Figure 6.1: Hyrax workers running on our Android smartphone cluster.

The performance of these phones is compared to the performance of a cluster of 10 AMD Opteron 1220 machines, each with 4 GB RAM, two Seagate Barracuda 7200.10 320 GB disks, and a Broadcom NetXtreme BCM5721 Gigabit Ethernet controller. Each node runs Debian GNU/Linux 4.0 (etch) with Linux kernel 2.6.18. In comparing the

Benchmark	**Input type(s)**	**Phone base input**	**Server base input**
Pi Estimator	Maps per host	3	27
Random Writer	Bytes per map, Maps per node	1 MB, 2	2500 MB, 2
Sort	Bytes per map, Maps per node	256 KB, 1	625 MB, 1
Grep	File size, Files per node	64 KB, 1	625 MB, 1
Word Count	Files per node, Files per node	32 KB, 1	141 MB, 1

Table 6.1: Benchmark input types and sizes per node.

performance of the phones and the servers, the relative performance capacity of each is taken into consideration.

6.1.2 Benchmarks

In our experiments, benchmarks that execute MapReduce jobs are run on Hyrax (in the case of phones) and Hadoop (in the case of servers). These benchmarks are Sort, Random Writer, Pi Estimator, Grep, and Word Count, all of which are derived from the Hadoop examples. The input size is scaled to be proportional to the size of the cluster and a number of maps is specified such that each node will be assigned some work.

Larger input sizes are used when running benchmarks on servers in order to compensate for the differences in CPU speed and bandwidth between servers and phones (determined in §6.2). For each benchmark on each hardware platform, a base input size that makes the benchmark last for around one minute is chosen. These input sizes and their types are summarized in Table 6.1. Note that these sizes per-node; the base input size is multiplied by the number of nodes to determine the total input size.

In the Random Writer benchmark, data is generated on each phone. In the Sort benchmark, sortable data is generated on each phone using Random Writer, and then this data is sorted using Hadoop's Sort example. Sort has a trivial map phase which just relays inputs to the reduce phase, taking advantage of the fact that Hadoop sorts the intermediate keys generated by map tasks. In the Pi Estimator benchmark, Hadoop's Pi Estimator example, which uses a Monte Carlo method to estimate the value of π, is executed, with a number of maps proportional to the number of nodes. The Grep benchmark places a set of large text files on HDFS and then searches for a word within them using Hadoop's Grep example. The Word Count benchmark places the same text data on HDFS and computes the number of occurrences of each word.

A control benchmark that runs Hyrax for 60 seconds without executing any jobs is

Benchmark	Initialization phase	Execution phase
Random Writer	None	Random Writer job
Sort	Generate files using Random Writer	Sort job on generated files
Grep	Generate text files, push to HDFS	Grep job on these files
Word Count	Generate text files, push to HDFS	Word count job on these files
Pi Estimator	None	Pi estimator job
Control	None	Sleep for 60 seconds

Table 6.2: Benchmark initialization and execution phases.

also run. The data collected in this benchmark can be used to account for the effects of Android's background processes and the overhead of the DataNode, TaskTracker, and sensor data manager that run within Hyrax.

Each benchmark consists of an initialization phase and an execution phase. Only the beginning and end of the execution phase are recorded, and the resource usage outside of this range is ignored. The initialization and execution phases of each benchmark are given in Table 6.2.

6.1.3 Analysis tools

In addition to simply recording benchmark completion times, two tools are used to analyze the performance of Hyrax in depth: system resource usage logs and Hadoop log analysis.

System resource usage logs

In order to study the system resource usage of Hyrax, relevant information from `/proc`, including CPU usage, memory usage, disk I/O, and network I/O, is logged. These values are logged about twice per second. Figure 6.2 shows an example of system metrics collected on phones during a run of the Sort benchmark.

Log analysis for Hadoop performance visualization

Hadoop log parsing techniques from Tan et al. [2009], Tan et al. [2008] are used to extract detailed information about task execution in our benchmarks, including the timing and duration of each task or task component. These techniques use logs generated by Hadoop to infer the state of each node in the cluster.

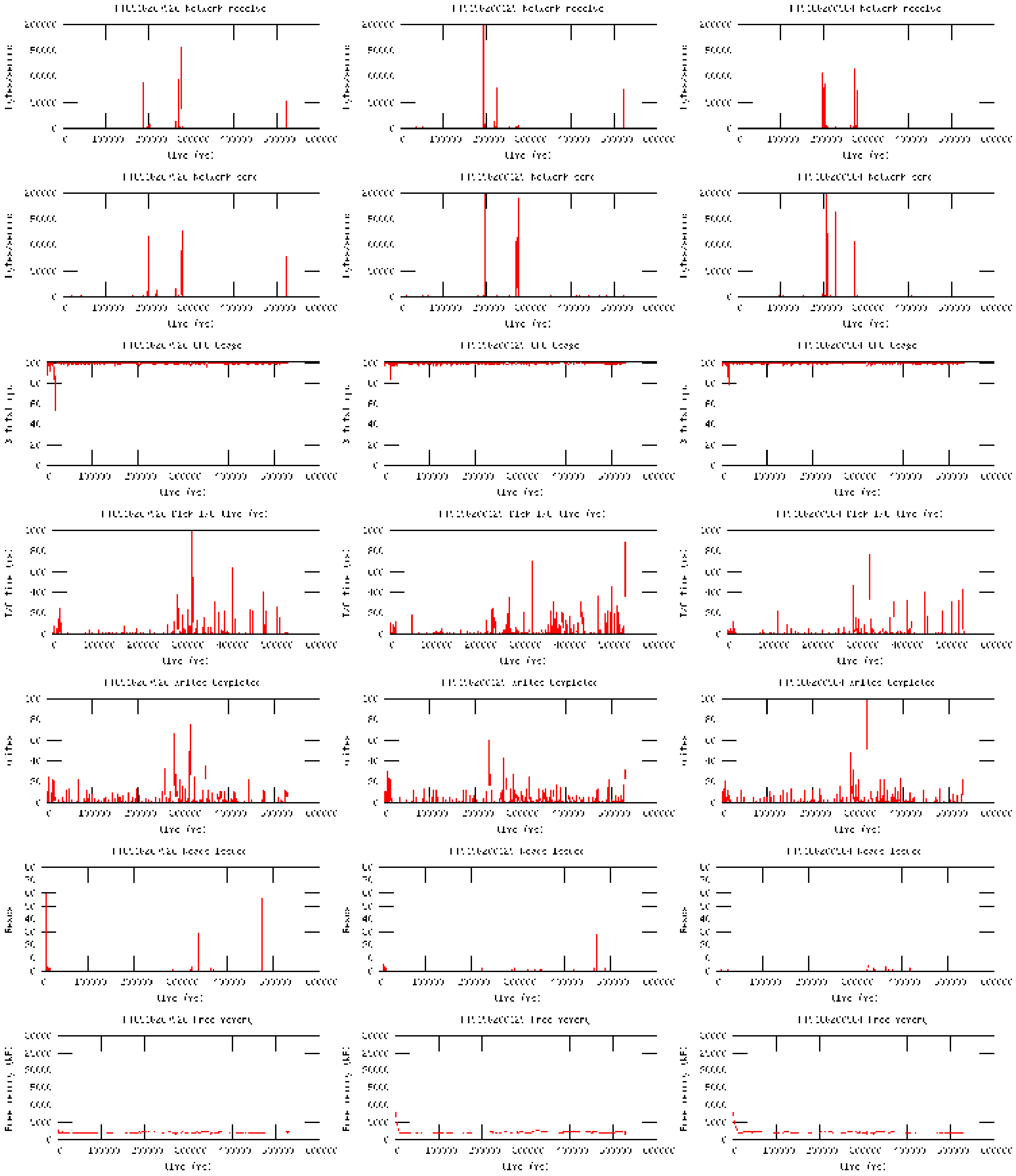


Figure 6.2: Example of system resource usage data. Network, CPU, disk, and memory usage metrics for Sort benchmark on 3 of 10 phones.

This information can be used to generate useful visualizations. For instance, "Swimlanes" plots show task progress as it unfolds in time. They also show how tasks running on different nodes progress in time. Hence, the swimlanes plots show MapReduce behavior in time and space. The x-axis denotes wall-clock time elapsed since the beginning of the job, and each horizontal line corresponds to the execution of a state (e.g., Map, Reduce) running in the marked time interval.

Figure 6.3 shows swimlanes plots for the Sort benchmark on 5 phones and on 5 servers. From this, we can immediately see the difference in individual task completion times between phones and servers. We also see which tasks were executed in parallel. To gain more insight, swimlanes and system metrics can be plotted on the same time axis to show how tasks affect system resource usage.

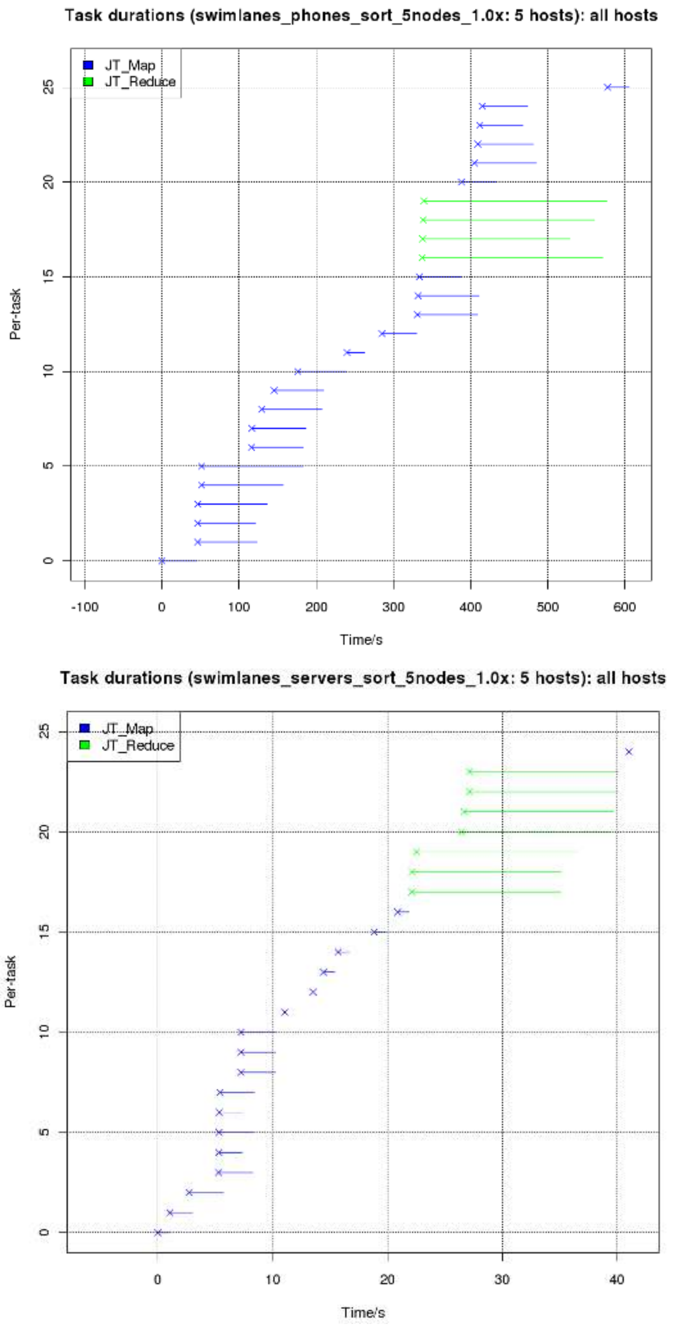
The log parsing system determines the amount of time spent in each Hadoop phase (map, reduce, shuffle, and sort). This information can be used to compare the relative amount of time spent in each phase between servers and phones. For example, Figure 6.4 shows the amount of time spent in each task or phase type for each node and for the cluster as a whole. This plot shows that the absolute time taken for the job on servers is much less. It also shows that the proportion of time spent on maps on the phone is larger than on the servers. Since Pi Estimator is a CPU-bound job, the graph implies that CPU performance on phones is worse in relation to its other resources than it is on servers.

6.2 Baseline performance of mobile devices vs. traditional servers

The inherent performance differences between the phones and servers in our testbed were investigated by comparing the speeds of four micro-benchmarks, each of which is bound by CPU, memory, disk, or network resources. The results are summarized in Table 6.3.

In the CPU benchmark, an empty loop is executed for some number of iterations. In the memory benchmark, a buffer is sequentially written to for some number of iterations and then sequentially read from. The times for these benchmarks were not significantly affected by the memory reads and writes, indicating that the memory benchmark was still CPU-bound on both the server and the phone. We concluded that the server is about 370 to 430 times faster than the phone for CPU-bound operations.

In the disk benchmark, data is sequentially written to the server's hard disk and to the phone's flash card. Of all the system capabilities that were tested, the server and the phone are closest in disk access speeds. The server is 7.6 times faster for writes and 30 times


Task durations (swimlanes_phones_sort_5nodes_1.0x: 5 hosts): all hosts

Task durations (swimlanes_servers_sort_5nodes_1.0x: 5 hosts): all hosts

Figure 6.3: Swimlanes visualization for Sort benchmark on 5 phones and on 5 servers sorted by task start time.

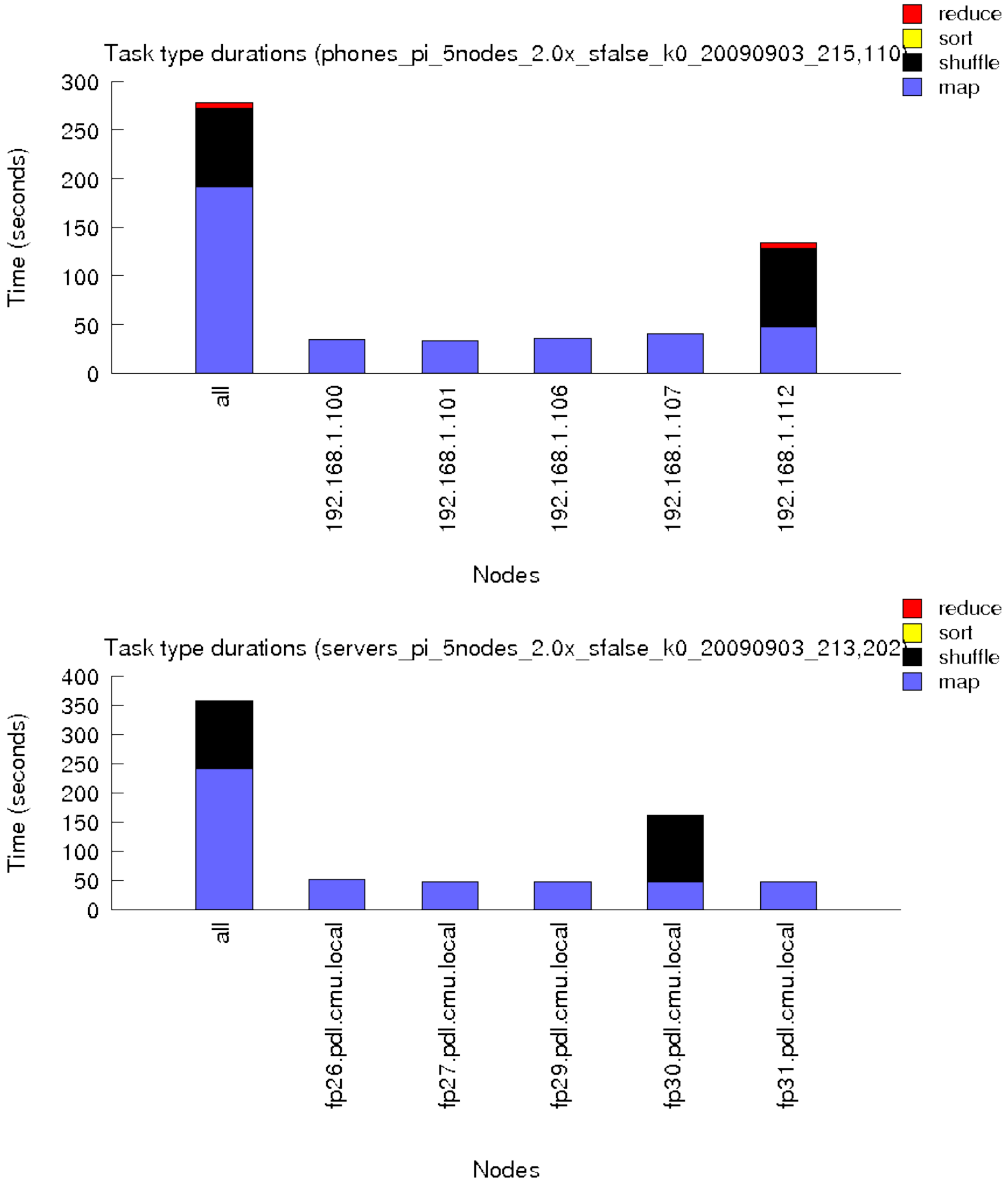


Figure 6.4: Total phase time bar graphs for 5 smartphones and for 5 servers running Pi Estimator.

Benchmark	G1 throughput (MB/s)	Server throughput (MB/s)	Server advantage
Memory write	12	4600	390x
Memory read	11	4500	430x
Disk write	8.7	66	7.6x
Disk read	15	460	30x
Network write	0.92	87	95x
Network read	0.64	86	140x
CPU	N/A	N/A	370x

Table 6.3: Android G1 and server performance results.

faster for reads.

In the network benchmark, socket servers running on the same hardware as on the device being tested are written to and read from. In the phone case, the benchmark program is run on one phone and socket servers are run on another phone connected to the same WiFi router. In the server case, the benchmark program is run on one machine and the socket servers are run on another machine on the same rack. The server outperformed the phone by a factor of 95 in write speeds and 140 in read speeds. We suspect that the asymmetry in read and write speeds between phones (0.92 MB/s for writes, 0.64 MB/s for reads) is an artifact of the Android libraries.

These performance differences inform how the results of Hadoop benchmarks, which make use different system resources to different extents, should be evaluated. In particular, we would expect the huge difference in CPU speeds (and therefore memory access speeds) between servers and phones to cause significant performance problems for Hyrax. Hadoop was written with the assumption of being disk or network-bound in most cases, and thus is not conservative in CPU usage.

Note that this poor CPU performance is a property of Android, not of mobile platforms in general. For instance, the iPhone has been shown to be about 100 times faster than the Android G1 in effective CPU cycles per second Occipital.

6.3 Network link properties

Another question relevant to the higher-level experiments, particularly the file sharing experiment, is how each link in the network contributes to data transfer times. In this experiment, the amount of time required to transfer varying amounts of data between elements of our testbed is studied.

6.3.1 Question

What are the relative speeds of data transfer between components of our testbed network?

6.3.2 Approach

The amount of time required to transfer data from a phone and another phone PP, from the server to a phone SP, and from a phone to a server PS are measured, varying the amount of data transferred. 7 iterations are performed for each link with each amount of data.

Using these results, the contribution (in seconds-per-byte) of the phone-router link PR, the router-phone link RP, the server-router link SR, and the router-server link RS to data transfer times can be estimated using the simplifying assumption that $PP = PR + RP$, $SP = SR + RP$, and $PS = PR + RS$. Note that it is not assumed that $SP = PS$. However, since only one server is present in testbed and there are not enough equations to solve for RS, SR, PR, and RP, it is also assumed that $SR = RS$.

This model does not account for the contributions of data transfer times within the devices, but it is precise enough for the purposes of our experiments.

6.3.3 Hypothesis

We expect that SR will be much smaller than RP and PR. In wireless networks, the "air time", i.e. the time taken to send data wirelessly, tends to be the bottleneck in data transfers.

We expect smaller transfers to be more costly in terms of bytes / second, particularly for the wireless link, because of the additional effects of overhead of establishing the connection.

6.3.4 Results

Figure 6.5 shows transfer time with respect to amount of data transferred for server to phone, phone to phone, and phone to server transfers. Figure 6.6 shows transfer time with respect to amount of data transferred for small transfers (up to 128 KB).

The inverse bandwidth of each link in Figure 6.5 for sufficiently large amounts of data

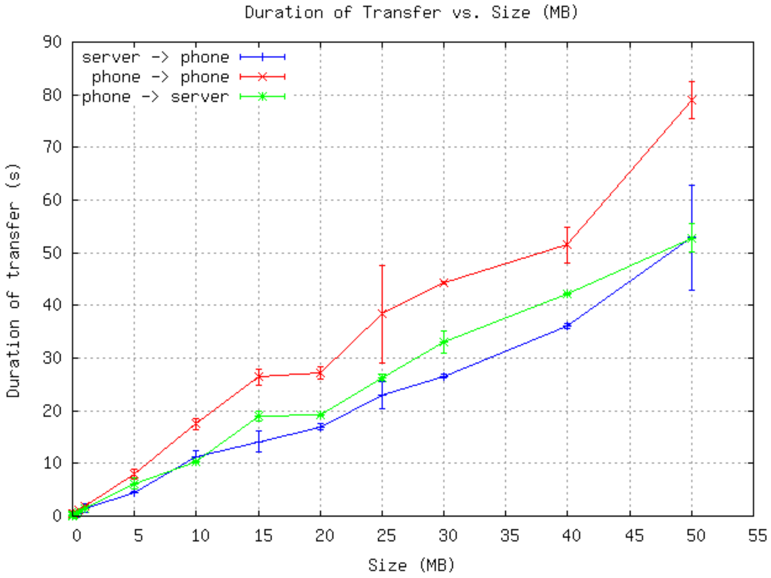


Figure 6.5: Network transfer time vs. size for each network path in testbed for large transfers.

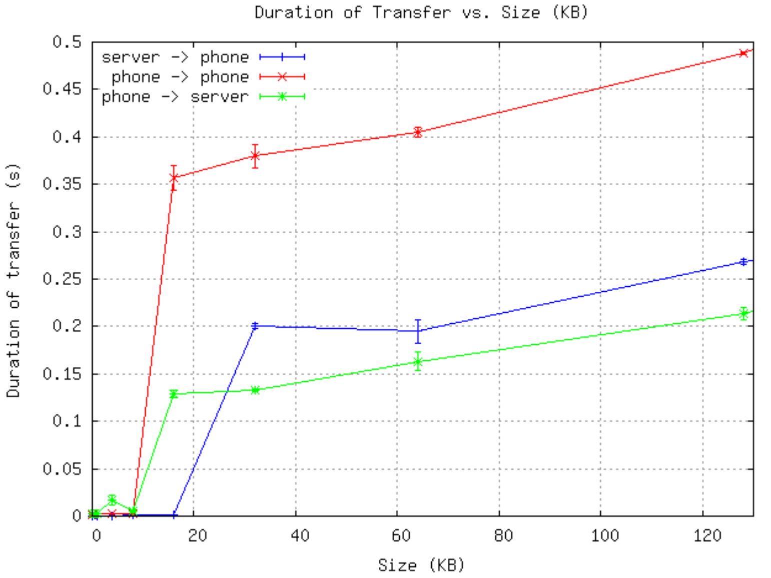


Figure 6.6: Network transfer time vs. size for each network path in testbed for small transfers.

is estimated by the slope of a linear fit to its curve. This yields:

$$PP = 1.47\,\mathrm{s/MB}$$

$$PS = 1.06\,\mathrm{s/MB}$$

$$SP = 0.97\,\mathrm{s/MB}$$

Solving the system of equations in the model yields:

$$PR = 0.78\,\mathrm{s/MB}$$

$$RP = 0.69\,\mathrm{s/MB}$$

$$SR = RS = 0.28\,\mathrm{s/MB}$$

6.3.5 Conclusions

According to Figure 6.5 and the estimated values of PS and SP, the server-phone link has a slight advantage over the phone-server link. This is most likely related to the performance difference between the server and the phones. The server-phone link and the phone-server link are significantly faster than the phone-phone link. According to the estimated values of PS, PP, and SP, the phone-phone link is about 50% slower than either of the phone-server or server-phone links.

The estimated values of PR, RP, and $SR = RS$ support our hypothesis that the wireless links contribute the most time to transfers. For phone-server and server-phone transfers, wireless links contribute about 70 to 75% of the total transfer times.

Figure 6.6 supports our hypothesis that for small transfers (up to about 128 KB), overhead contributes more to transfer time than the marginal cost of additional bytes.

6.4 Performance of Hadoop on mobile devices and traditional servers

According to Satyanarayanan [1996b], mobile elements are inherently poor relative to static elements because the additional weight, power, and size restrictions compared to static counterparts will always have a negative effect on performance and capacity. In this experiment, the effect of mobile resource constraints on Hadoop is quantified.

6.4.1 Question

The following questions are addressed in this experiment:

1. What effects do mobile resource constraints have on the performance of Hadoop?
2. In terms of input size and numbers of nodes, how does Hadoop scale on mobile devices compared to how it scales on servers?
3. What resources are bottlenecks for Hadoop on mobile hardware?

Approach

In order to answer these questions, MapReduce benchmarks were run on both Hyrax and Hadoop, varying the number of nodes and the size of the input data.

Each benchmark listed in §6.1.2 (Sort, Random Writer, Pi Estimator, Grep, Word Count, and control) was run on phones and servers varying the number of nodes in the cluster from 1 to 9 and using multiples of the base input data size of 0, 0.5, 1, 1.5, and 2. At least five iterations of each hardware, benchmark, nodes, and input data size configuration were performed. The replication factor was set to 2 in all cases. The Hadoop/Hyrax cluster was shut-down and reinitialized before each experiment.

For each hardware, number of nodes, and data size, the mean execution time, mean total component task times, and average total resource usage are computed. For each mean, a confidence interval is computed by multiplying the standard error by the t-distribution value (for $\alpha = 0.025$) corresponding to the number of samples. This is displayed as an error bar around each point. For a given set of N samples $x_0 \ldots x_{N-1}$, the standard error of the mean is computed as

$$SE_{\bar{x}} = \frac{s}{\sqrt{N}}$$

where s is the sample standard deviation:

$$s = \sqrt{\frac{1}{N-1}\sum_{i=0}^{N-1}(x_i - \bar{x})^2}$$

and $\bar{x}$ is the arithmetic mean of $x_0 \ldots x_{N-1}$. Note that some confidence intervals are very small because of low variance and thus their corresponding error bars are not clearly visible in the plots.

Because such vastly different input sizes are used between servers and phones (see Table 6.1), the execution times cannot be directly compared between the two hardware platforms. Since the input sizes for the phone benchmarks are so much smaller, the overhead of setting up tasks and transferring data have a much more significant effect than they do on servers. At best, the way in which these times vary with parameters such as number of nodes and input size can be compared.

6.4.2 Hypothesis

Considering the observations made in §6.2, we expect MapReduce jobs to be much slower on mobile phones compared to on servers. We expect phones to be bottlenecked on CPU (and thus memory operations) most of the time because of the inherent deficiency of these resources on the phones in our testbed. This would cause jobs to spend more time in CPU-intensive tasks (maps, reduces) relative to other the task types. It would also be manifested in the CPU usage system metric.

Since there are no fundamental differences in the architecture of Hadoop and Hyrax, Hyrax should be able to scale linearly, at worst, with input data size and number of nodes. We do not expect any superlinear increases in resource usage or job/task completion times with increasing numbers of nodes and input sizes.

According to Amdahl's law Amdahl [1967], the change in execution and task times of a benchmark with the number of nodes should depend on what portion of the benchmark can be executed in parallel. Amdahl's law states that the maximum speed-up S_n by using n nodes is

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

where P is the portion of the task that can be executed in parallel.

We can apply this to model the effect of the number of nodes on execution time and task times. Assuming a fixed input size,

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

where E_n is the execution time when the number of nodes is n. Recall that in our experiments the input size I_n is always proportional to n:

$$I_n = nI_1$$

Therefore it makes sense to work with E_n/I_n, the execution time per unit of input:

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

Hence:

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

Therefore we expect the execution time to increase approximately linearly with a slope of $(1-P)E_1$ where P depends on how much of the task is executed (independently) in parallel. Figure 6.7 shows the expected E_n vs. n curve with varying values of P. Differences in the slope of the execution time and task times vs. number of nodes plots would indicate different levels of parallelism between phones and servers.

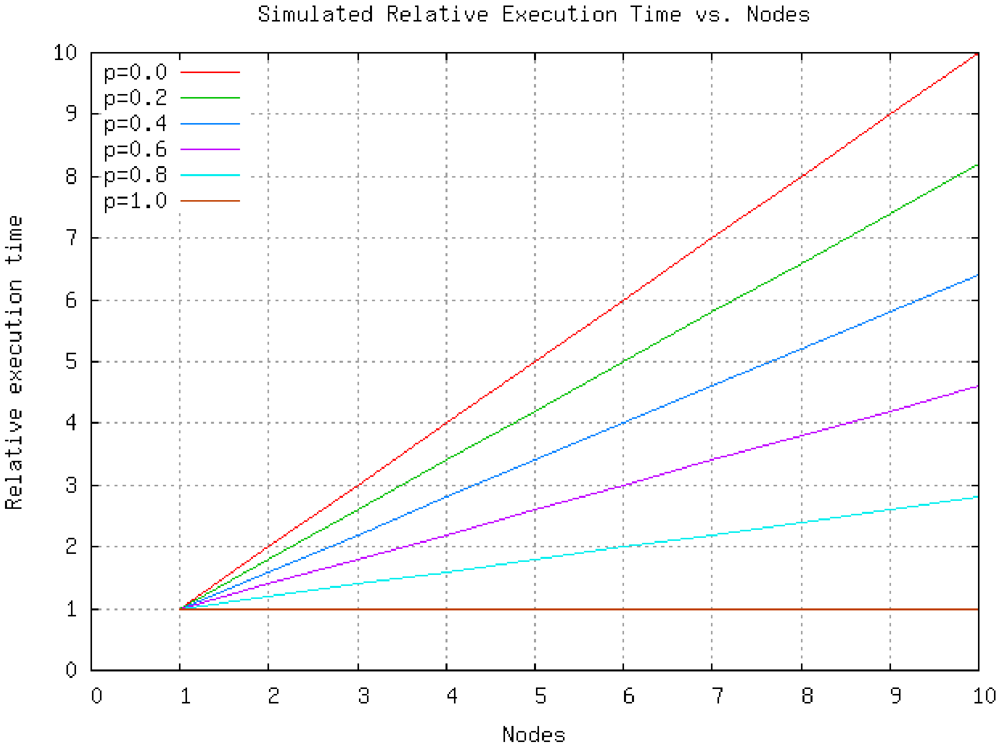


Figure 6.7: Simulated relative benchmark execution time vs. number of nodes for varying levels of parallelization.

6.4.3 Results

Execution time

The execution time E_n of a benchmark is the total time spent in the benchmark's execution phase on a cluster of n nodes. This gives a very course-grained view of benchmark performance.

In general, E_n increases with n as predicted by our model. There is usually a relatively large jump between $n = 1$ and $n = 2$. The rate of increase of E_n is very similar between phones and servers in all cases, indicating that the hardware does not have much effect on the parallelism of a given benchmark.

The effect of the input size is reflected to a much larger extent in servers than in phones. For instance, in the Sort benchmark (Figure 6.8), execution time more than doubles from 1.0x to 2.0x on servers, whereas it remains nearly the same for phones. This is probably because the input sizes in the phone benchmarks are so small by comparison, so overhead has a much larger effect. A similar effect is observed in Random Writer (Figure 6.9) and Word Count (Figure 6.10).

Task times

Task information was extracted from Hadoop logs using the log analysis system described in 6.1.3. For each benchmark run, the total time for each task type ($T_{\text{map},n}$, $T_{\text{shuffle},n}$, $T_{\text{sort},n}$, $T_{\text{reduce},n}$, where n is the number of nodes) is computed. Note that the total of the task times $T_{*,n}$ does not correspond to execution time since time is over-counted for parallel tasks, and parts of the execution may not involve any of these tasks.

Figure 6.11 shows normalized task time breakdowns for phones and servers vs. number of nodes n and input size s. For phones, $T_{\text{map},n}$ accounts for less of $T_{*,n}$ as n increases, while $T_{\text{shuffle},n}$ accounts for less of $T_{*,n}$. However, the rate at which this portion increases decreases with n. Figure 6.12 shows that both $T_{\text{map},n}$ and $T_{\text{shuffle},n}$ increase with n for phones.

$T_{\text{map},n}$ and $T_{\text{shuffle},n}$ account for larger portions of $T_{*,n}$ in phones than in servers, while $T_{\text{reduce},n}$ accounts for a larger portion of $T_{*,n}$ in servers than in phones. This indicates that phones are taking much longer to complete map tasks, which is probably related to the limited amount of memory available for buffering the output of map tasks or simply the relatively poor CPU performance of the G1. Recall that the Sort benchmark uses a trivial map function that simply forwards the input.

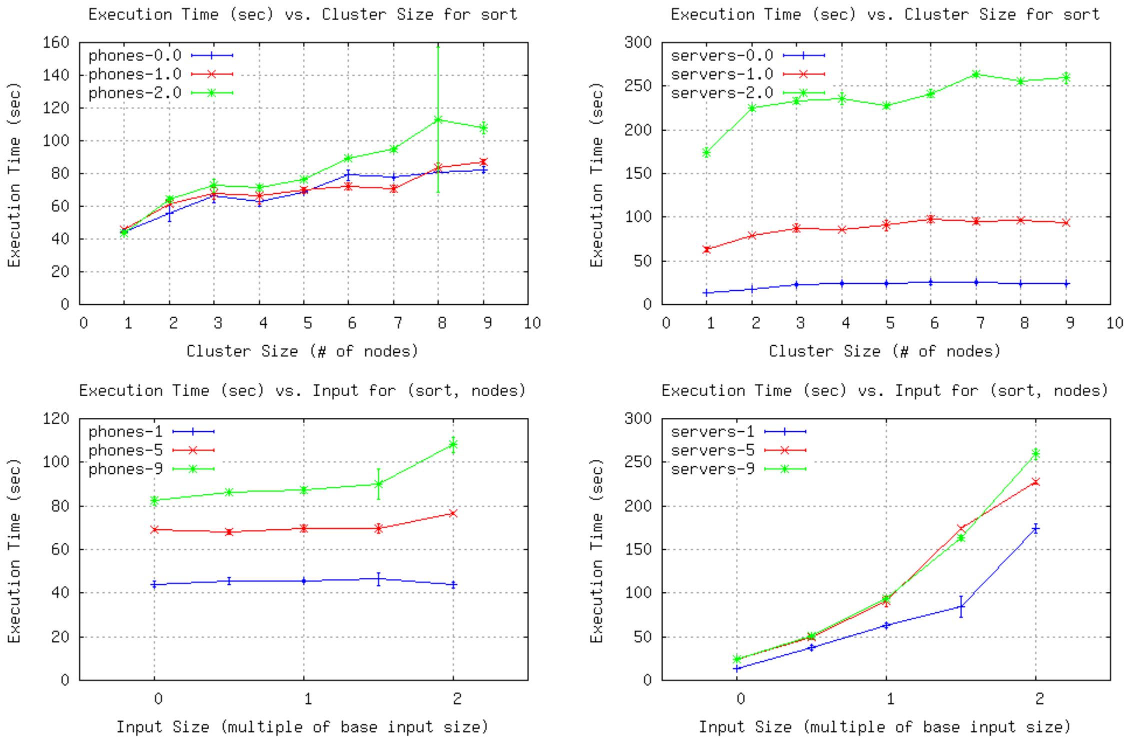


Figure 6.8: Execution time vs. number of nodes (top) and input size (bottom) for phones (left) and servers (right), Sort benchmark

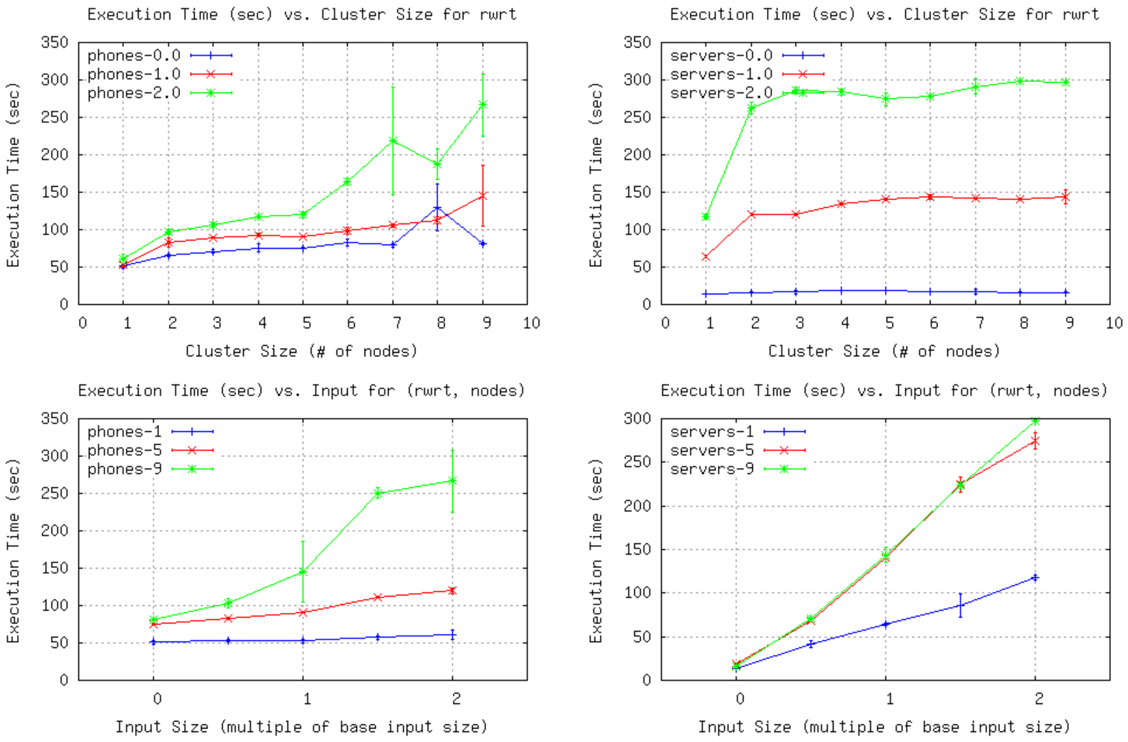


Figure 6.9: Execution time vs. number of nodes (top) and input size (bottom) for phones (left) and servers (right), Random Writer benchmark

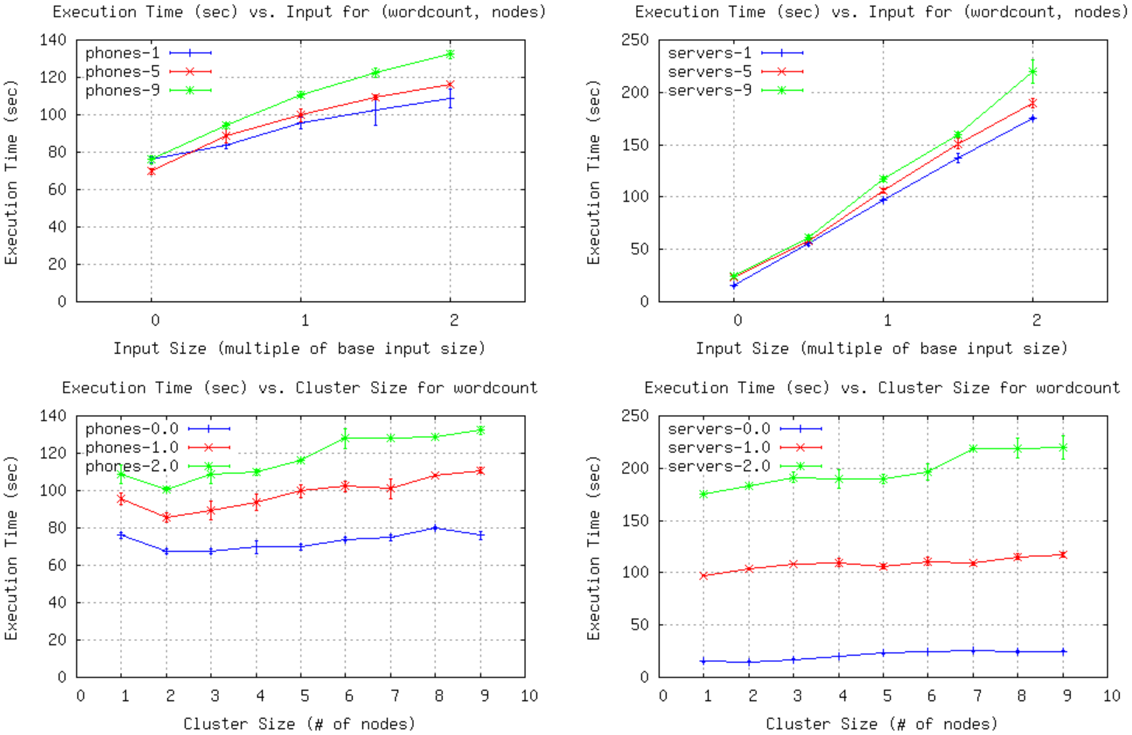


Figure 6.10: Execution time vs. number of nodes (top) and input size (bottom) for phones (left) and servers (right), Word Count benchmark

Figure 6.13 shows absolute task times vs. input size. For phones, $T_{*,n}$ does not vary much with the input size, whereas input size does have a large effect on $T_{*,n}$ for servers. This is probably related to the absolute differences in input sizes used between servers and phones.

Figure 6.14 shows normalized task time breakdowns for the Word Count benchmark. In Word Count, in contrast to Sort, for both servers and phones, $T_{\mathrm{map},n}$'s portion of $T_{*,n}$ increases with n. The only difference between servers and phones in this case is that in phones $T_{\mathrm{reduce},n}$ and $T_{\mathrm{sort},n}$ account for a significant portion of $T_{*,n}$, whereas they are practically insignificant for servers.

Resource usage

Using resource usage logs, we computed, for each experiment, total bytes sent, total bytes received, total disk io, total disk writes, total disk reads, and average CPU utilization.

Total bytes sent and received is very consistent on both phones and servers. Figure 6.15 shows the total bytes sent and received for the Sort benchmark. These metrics increase linearly for both phones and servers.

Figure 6.16 shows average CPU utilization across all nodes vs. number of nodes for Sort and Pi Estimator. For a given input size, average CPU is always lower on servers than on phones. For both servers and phones, average CPU usage decreases with the number of nodes.

Figure 6.17 shows disk reads, disk writes, and disk I/O time for the Word Count benchmark. Total disk reads and total disk I/O are highly variable and don't exhibit clear trends. Total disk writes is less variable and tends to increase with input size and number of nodes for both phones and servers.

6.4.4 Conclusions

These results show several differences and similarities between phones and servers.

There is a huge difference in the amount of data that phones and servers can process in a given amount of time. Servers were able to process 1000x to 5000x more data than phones in the same amount of time. Hadoop appears to have a huge base cost when run on Android, making it very slow and costly to process even small amounts of data. Reducing this base cost is perhaps the most important challenge in that must be addressed Hyrax before it can be used for real-world applications. Based on this performance difference,

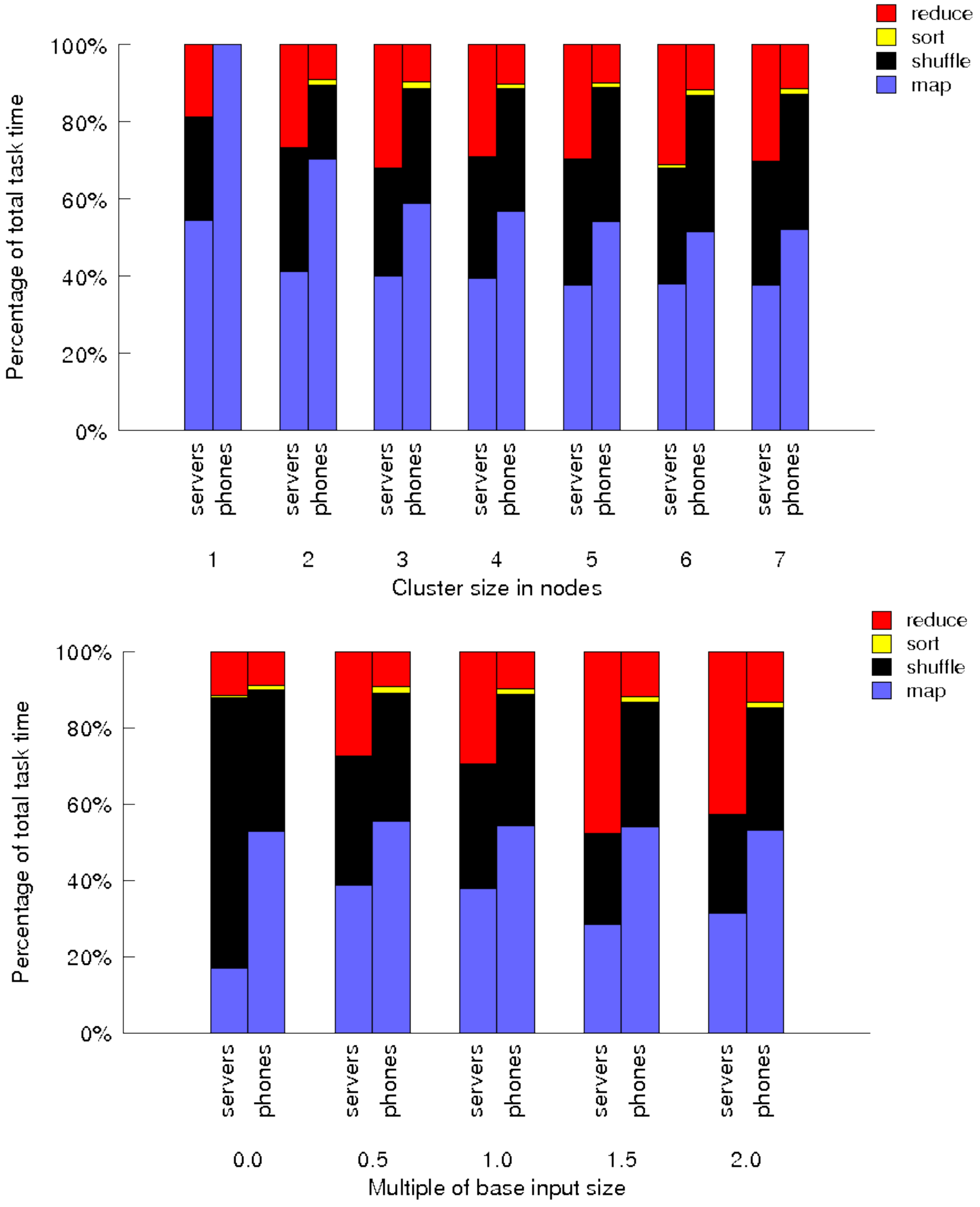


Figure 6.11: Normalized task time breakdown for servers and phones vs. number of nodes (top, 1.0x input size) and input size (bottom, 5 nodes) for Sort benchmark.

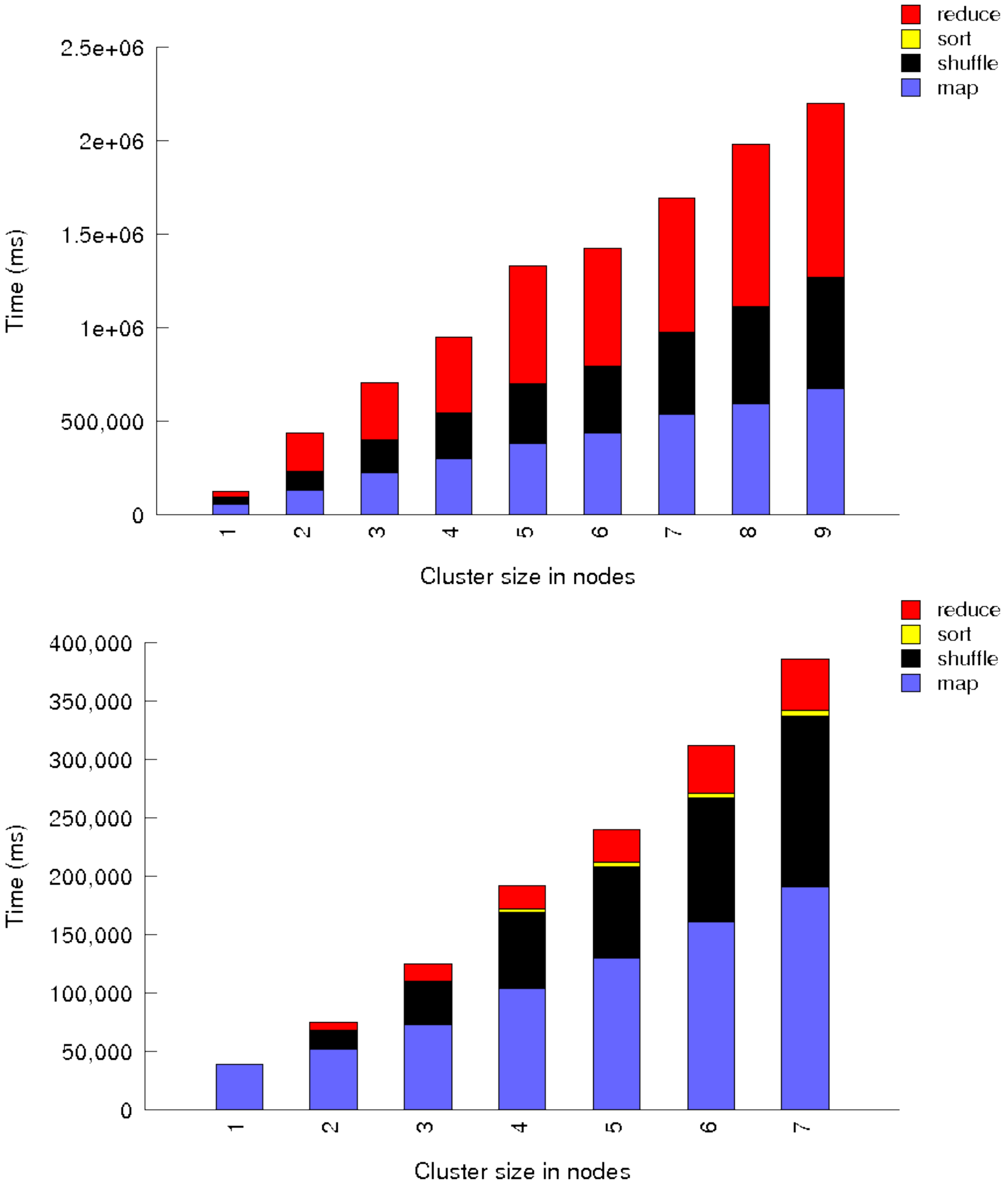


Figure 6.12: Absolute task time breakdown for servers (top) and phones (bottom) vs. number of nodes for Sort benchmark, 1.5x base input size.

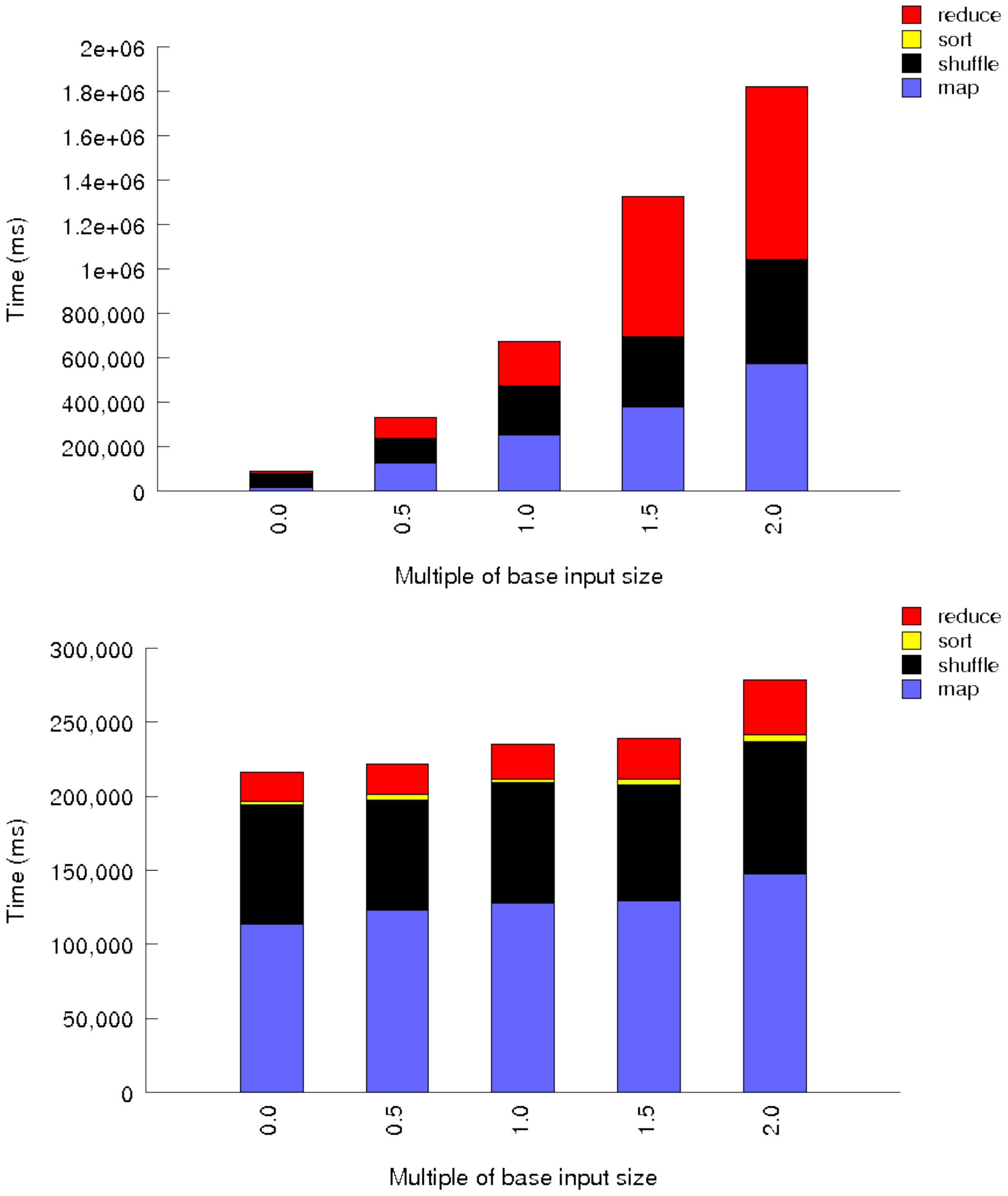


Figure 6.13: Absolute task time breakdown for servers (top) and phones (bottom) vs. input size for Sort, 5 nodes.

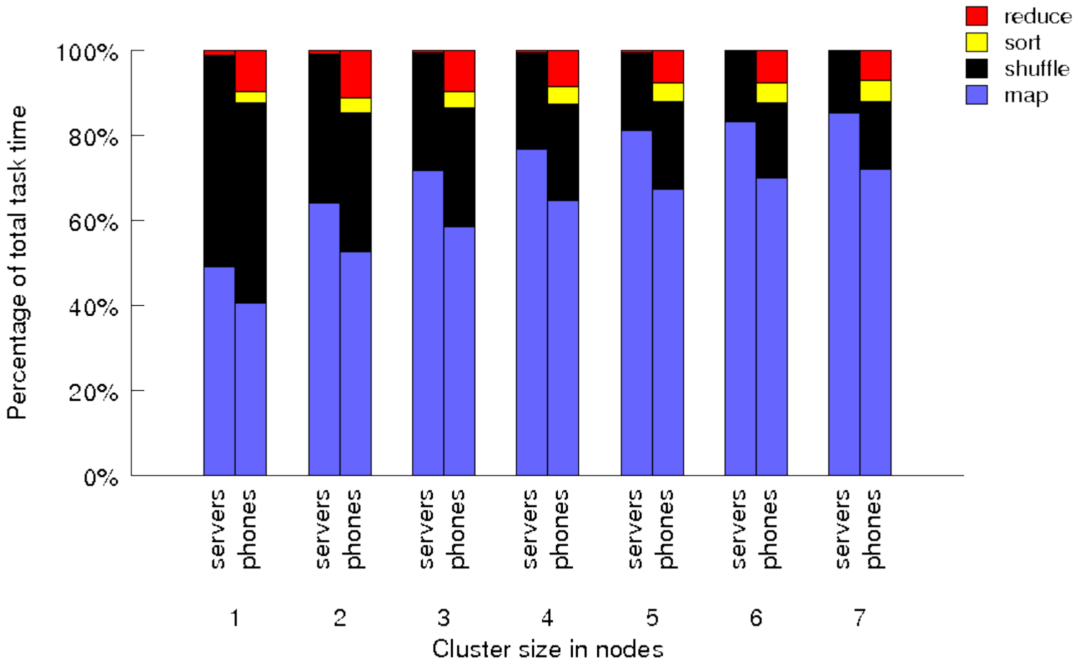


Figure 6.14: Normalized task time breakdown for servers and phones vs. number of nodes (1.0x input size) for Word Count.

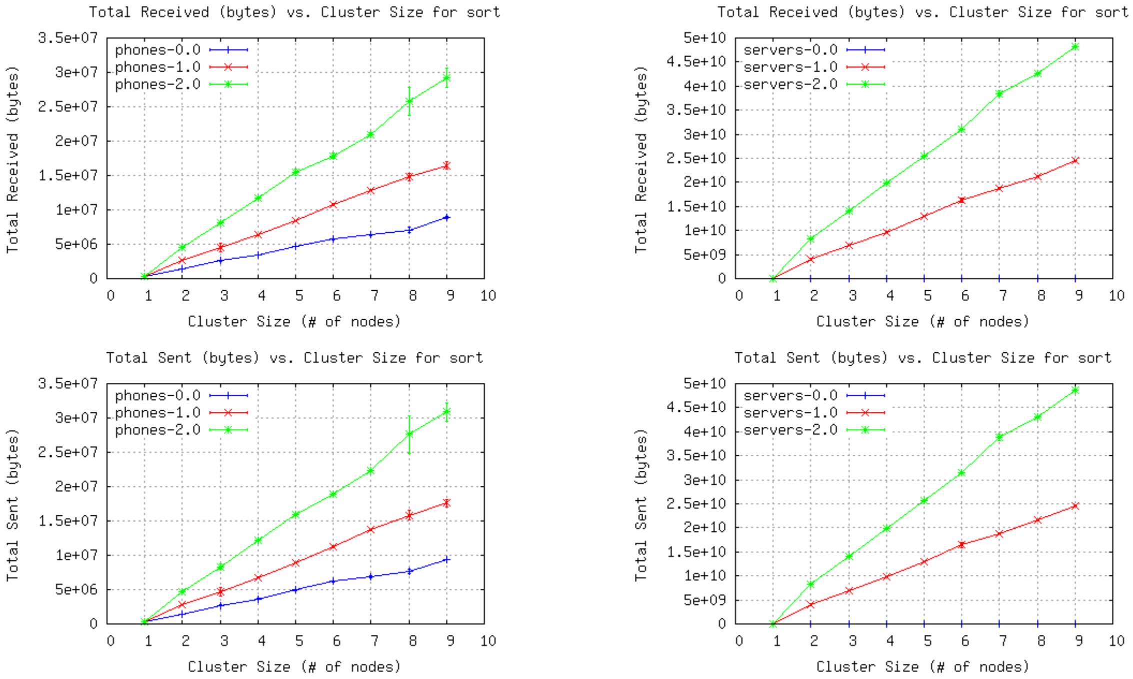
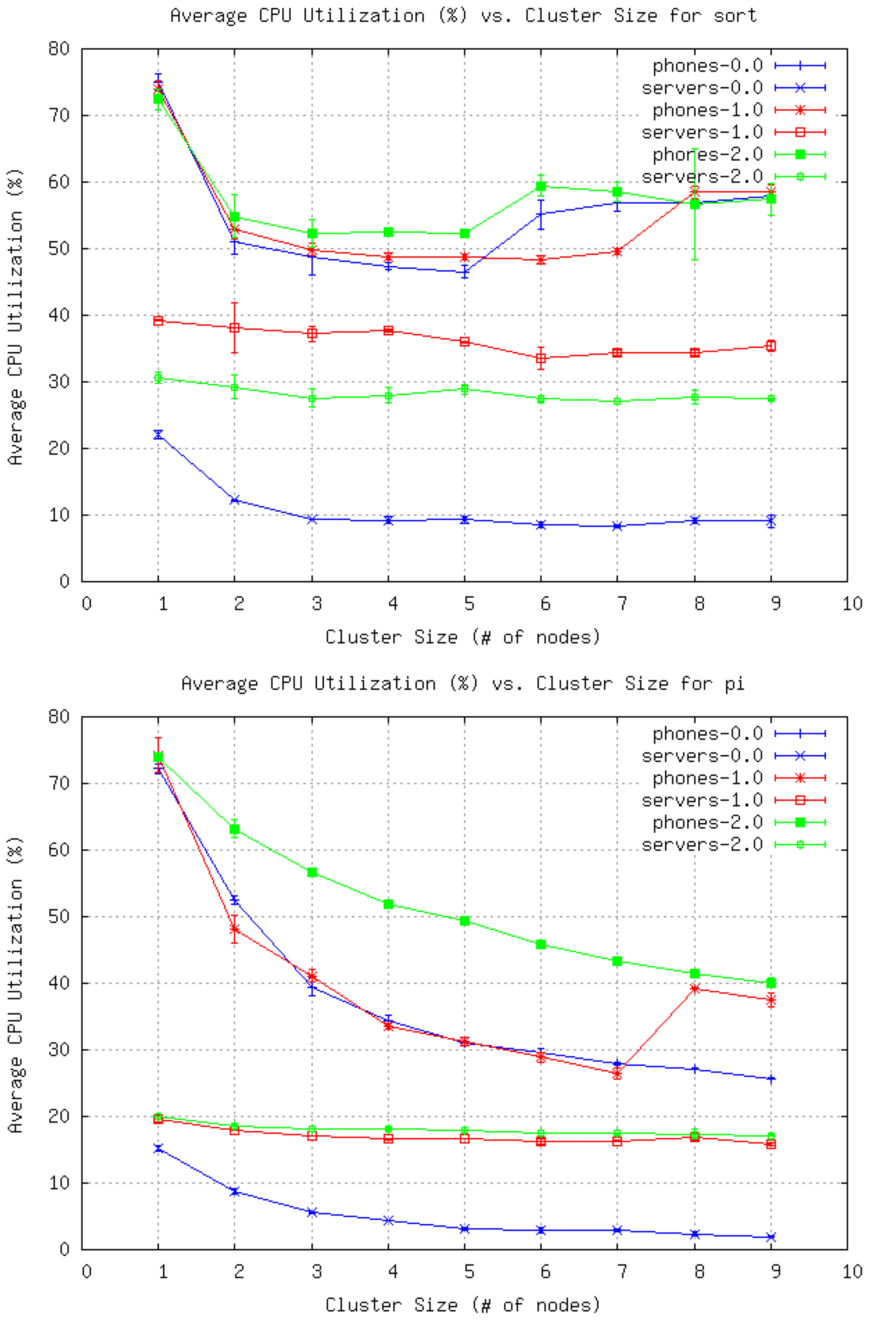


Figure 6.15: Total bytes received (top) and sent (bottom) vs. number of nodes for servers and phones, Sort benchmark. The server plot for input size 0 is nearly zero for all numbers of nodes.


Average CPU Utilization (%) vs. Cluster Size for sort

Average CPU Utilization (%) vs. Cluster Size for pi

Figure 6.16: Average CPU usage vs. number of nodes for servers and phones, Sort (top) and Pi Estimtor (bottom) benchmarks.

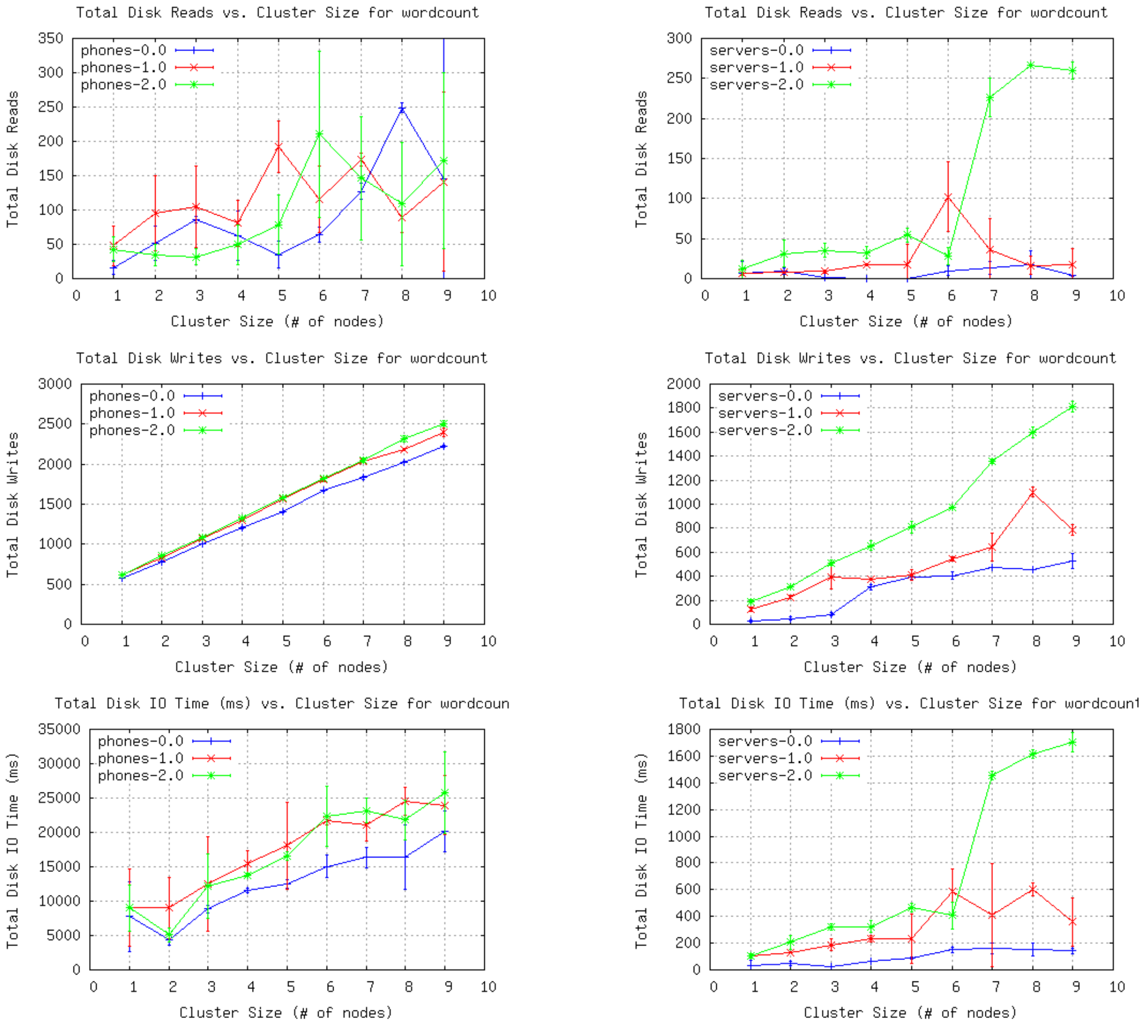


Figure 6.17: Word Count benchmark disk reads (top), disk writes (middle), and disk I/O time (bottom) vs. number of nodes for phones (left) and servers (right).

in the current state of Hyrax, it would take upwards of 1000 Android G1s to achieve the performance of a single server, assuming a network powerful enough to handle all of these wireless connections and a perfectly parallel workload.

Note that this performance problem is imposed at least partially by the artificial memory limitation that Android imposes on its applications. If even a few more MB of memory were available to the Hyrax worker, much more data could be processed without swapping to files, which is a huge performance burden on mobile devices.

The execution time required to complete a job increased at similar rates with n on both phones and servers. This shows that Hadoop and Hyrax scale similarly for $1 \leq n \leq 10$ in terms of job completion. There was no clear difference in the variance of execution times between phones and servers, suggesting that the amount of time required to complete a job is similarly predictable on both.

Overhead costs had a much larger effect on E_n on phones than on servers. This is either because the input sizes tested on phones were not signficantly different from each other, because the overhead of setting up and shuffling data among tasks is higher on phones, or a combination of these factors. It is difficult to tell whether Hyrax and Hadoop scale similarly in terms of data because the effects of overhead were so high on phones.

Mapping and shuffling account for a larger portion of task times on phones than on servers. In the case of maps, this is most likely related to the CPU and memory limitations of the phones. In the case of shuffles, this is probably caused by the difference in network speed between wireless and wired connections (shown to be about 100-150x in 6.2).

Changes in resource usage with n are similar on phones and servers. Network sends, network receives, and disk writes increase linearly with n. There are no significant patterns in disk reads and disk I/O times. Average CPU utilization is higher on phones, and CPU utilization decreases with n. Overall, Hyrax scales similarly to Hadoop with the number of nodes.

Considering the differences in CPU utilization and the amount of time spent on map tasks, it appears that CPU and memory are the biggest resource bottlenecks for Hyrax on the Android platform. The memory limitation is artificial and could be alleviated by modifying a constant in the Android source code. The CPU limitation is a more fundamental problem, probably related to using a non-optimizing VM instead of directly using hardware to execute code.

6.5 Handling network changes

Satyanarayanan [1996b] also notes that mobile devices are more susceptible to loss and damage, and mobile connectivity is highly variable in performance and reliability, and there is often no network available. As a result of variations in network connectivity and in some cases loss or damage, phones are expected to intermittently drop out of the network. A mobile-cloud computing system should handle devices departing from the network.

In Hyrax, when a node departs the network, its data blocks and intermediate MapReduce results go with it. Given how frequently node departure occurs in a mobile device network, it is important to determine the extent to which Hyrax can recover from it.

6.5.1 Question

In this experiment addresses the question: to what extent does Hyrax tolerate node departure? In other words, under what conditions does Hyrax succeed or fail to complete tasks when nodes leave the network?

6.5.2 Approach

These questions are addressed by running the benchmarks in §6.1.2 and killing the DataNode and TaskTracker instances on k random nodes 30 seconds after the beginning of job execution. For each benchmark, the number of nodes n is varied from 1 to 7, k is varied from 0 to 3, and the replication factor r is varied from 1 to 3. $k = n$ is not tested because killing all n nodes would definitely cause the job to fail. A success is defined to be a case where the benchmark completes, and a failure is defined to be a case where the benchmark fails. Each configuration is tested 5 times. The success rate of a given (n, k, r) is the number of successes over the total number of attempts.

6.5.3 Hypothesis

Hadoop is designed to tolerate node failures. Block replication decreases the likelihood of total block loss when a node leaves the network. In an HDFS cluster with replication factor r, in order for a block of data to be lost completely, all r nodes hosting its replicas must leave the network within a small amount of time. This amount of time is related to how often the NameNode expects pulse messages from the DataNodes and how long it takes to transfer a block of data between two nodes.

Nodes / Kills	0	1	2	3
1	1.0,1.0,1.0	N/A	N/A	N/A
2	1.0,1.0,1.0	1.0, 0.0, 0.0	N/A	N/A
3	1.0,1.0,1.0	0.8, 0.8, 0.0	0.8, 0.0, 0.0	N/A
4	1.0,1.0,1.0	1.0, 0.8, 1.0	0.8, 0.8, 0.0	0.8, 0.0, 0.0
5	1.0,1.0,1.0	1.0, 1.0, 1.0	0.2, 0.6, 0.6	0.6, 0.8, 0.0
6	1.0,1.0,1.0	1.0, 0.8, 1.0	1.0, 1.0, 0.8	0.8, 0.8, 0.0
7	1.0,1.0,1.0	0.8, 1.0, 0.8	1.0, 1.0, 0.8	0.6, 0.8, 0.4

Table 6.4: Node departure success rates for Random Writer benchmark. Each cell contains the success rates for $r = 1$, $r = 2$, and $r = 3$ in that order. Success rates where $k \geq r$ show shown in red.

When $k < r$, assuming no other problems in the system, it is impossible for data blocks to be lost completely. The NameNode is expected to recognize when a block is missing and replicate it accordingly. The JobTracker should identify and re-execute tasks that have taken too long or for which fetching the intermediate results has failed. Therefore we expect jobs to succeed when $k < r$. However, they may take significantly longer than jobs for which tasks don't fail because of the time required to identify and re-execute failed tasks and re-replicate blocks.

When $k \geq r$, it is more likely for a job to fail since the data that it is supposed to process may be completely lost. There is no way to re-generate the intermediate outputs of map tasks for which the input blocks are lost. Furthermore, since each block tends to be processed on the node where it is stored, it is likely that intermediate results will be lost if a node where the block is stored is lost. Therefore we expect jobs to fail often when $k \geq r$, especially when n is not much larger than k.

6.5.4 Results

The results are summarized in Tables 6.4, 6.7, 6.6, and 6.5. In these tables, each cell contains the success rates (successful attempts / total attempts) for $r = 1$, $r = 2$, $r = 3$ (in that order). Entries that were expected to have a low success rate (when $k \geq r$) are marked in red. Low "red" values are expected, but higher "red" values are good. "Black" values that are less than 1.0 probably indicate flaws in Hadoop.

Nodes / Kills	0	1	2	3
1	1.0,1.0,1.0	N/A	N/A	N/A
2	1.0,1.0,1.0	0.4, 0.0, 0.0	N/A	N/A
3	1.0,1.0,1.0	0.0, 0.6, 0.0	0.0, 0.0, 0.0	N/A
4	1.0,1.0,1.0	0.2, 0.2, 0.2	0.0, 0.0, 0.0	0.2, 0.0, 0.0
5	1.0,1.0,1.0	0.2, 1.0, 0.4	0.0, 0.0, 0.0	0.0, 0.0, 0.0
6	1.0,1.0,1.0	0.2, 1.0, 0.6	0.0, 0.6, 0.6	0.0, 0.0, 0.0
7	1.0,1.0,1.0	0.2, 1.0, 1.0	0.0, 0.8, 0.8	0.0, 0.2, 0.0

Table 6.5: Node departure success rates for Grep benchmark. Each cell contains the success rates for $r = 1$, $r = 2$, and $r = 3$ in that order. Success rates where $k \geq r$ show shown in red.

Nodes / Kills	0	1	2	3
1	1.0,1.0,1.0	N/A	N/A	N/A
2	1.0,1.0,1.0	0.0, 0.2, 0.0	N/A	N/A
3	1.0,1.0,1.0	0.4, 1.0, 0.0	0.0, 0.0, 0.0	N/A
4	1.0,1.0,1.0	0.2, 1.0, 1.0	0.0, 0.4, 0.0	0.0, 0.0, 0.0
5	1.0,1.0,1.0	0.4, 1.0, 1.0	0.0, 0.8, 1.0	0.0, 0.2, 0.0
6	1.0,1.0,1.0	0.2, 1.0, 1.0	0.2, 0.6, 1.0	0.0, 0.2, 0.6
7	1.0,1.0,1.0	0.2, 1.0, 0.8	0.0, 0.6, 1.0	0.0, 0.2, 0.8

Table 6.6: Node departure success rates for Word Count benchmark. Each cell contains the success rates for $r = 1$, $r = 2$, and $r = 3$ in that order. Success rates where $k \geq r$ show shown in red.

Nodes / Kills	0	1	2	3
1	1.0,1.0,1.0	N/A	N/A	N/A
2	1.0,1.0,1.0	0.0, 0.2, 0.6	N/A	N/A
3	1.0,1.0,1.0	0.0, 1.0, 0.6	0.0, 0.0, 0.0	N/A
4	1.0,1.0,1.0	0.0, 1.0, 1.0	0.0, 0.4, 0.8	0.0, 0.0, 0.0
5	1.0,1.0,1.0	0.0, 1.0, 1.0	0.0, 0.8, 1.0	0.0, 0.0, 0.0
6	1.0,1.0,1.0	0.0, 1.0, 1.0	0.0, 0.6, 0.8	0.0, 0.4, 0.4
7	1.0,1.0,1.0	0.0, 1.0, 1.0	0.0, 0.8, 1.0	0.0, 0.2, 0.0

Table 6.7: Node departure success rates for Sort benchmark. Each cell contains the success rates for $r = 1$, $r = 2$, and $r = 3$ in that order. Success rates where $k \geq r$ show shown in red.

6.5.5 Conclusions

As expected, there was a tendency for success rates to increase with n and decrease with k. However, there were many cases where benchmarks failed even when $k < r$. This may have to do with Hadoop failing to detect missing nodes, reassigning tasks to the same node even after the first failure. On the other hand, in the Word Count benchmark, there were many cases where the job succeeded even when $k \geq r$.

There are several cases in which success rates for a given k and r did not increase monotonically with n or r. These artifacts may indicate problems in Hadoop's replication or task-reassignment algorithms.

Overall, Hyrax recovers rather effectively from faults in Sort and Word Count, but not quite as well from faults in Grep and Random Writer. However, even in Grep and Random Writer, success rates increase with n, suggesting that better fault-tolerance would be possible with more nodes.

6.6 File sharing

One of the motivations for Hyrax is to avoid using remote services to share data when the data is available on devices in the local network to begin with. This experiment evaluates the performance of file sharing using HDFS vs. offloading data to a remote server.

6.6.1 Question

In this experiment, we ask: how does the performance of file sharing among mobile devices using Hyrax compare to file sharing using a remote server?

6.6.2 Approach

We publish a file from one node in the network, and then concurrently retrieve this file on all other nodes. In one case, which we label U, publishing is performed by uploading to a server outside of the local network, and retrieval is performed by downloading from this server. In the other case, publishing is performed by putting the file on HDFS, and retrieval is performed by pulling the file from HDFS (see Figure 6.18). We vary the number of nodes n, the size f of the file, and the replication factor r of HDFS. This case is labeled

$H(r)$. In both U and $H(r)$, the latency of the publishing phase, the latency of the retrieval phase, total bytes sent, and total bytes received are considered.

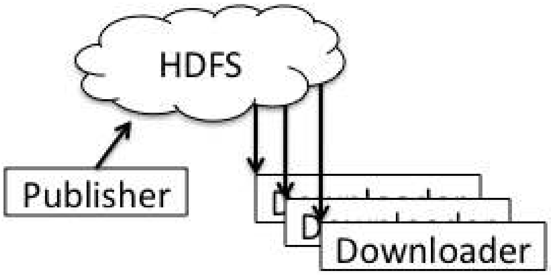


Figure 6.18: File sharing experiment diagram for the HDFS case.

6.6.3 Hypothesis

In U, we expect the latency of the publishing phase to depend only on f. We expect the latency of the downloading phase for U to increase with n because of contention for bandwidth from the server.

We expect the latency of the publishing phase to increase with r because of the additional copying that must be performed to replicate the new data blocks. For example, for $r = 1$, the publishing phase should be very fast and not use the network since the block will be stored on the local device. We expect the latency of the retrieval phase to decrease as r increases because of additional parallelism in block serving and availability of replicas locally on $r - 1$ of the downloaders.

The latency of a file transfer is closely related to the number of bytes sent through the network, especially across wireless links (as demonstrated in §6.3). Furthermore, sending and receiving bytes consumes battery. Therefore it is important to consider the total bytes sent and received in the publishing and retrieval phases of this experiment. Note that since a router is used in our testbed, a "receive" is equivalent to a send from the router to the receiver, and a "send" is a send from the sending device to the router.

In the case of $H(r)$, the total bytes sent by phones during the publishing phase $PS_{H(r)}$ should be

$$PS_{H(r)} = (\min(r, n) - 1)f$$

The total number of bytes received during the publishing phase $PR_{H(r)}$ should be equal to $PS_{H(r)}$ since all sent data is received by devices in the network. When we upload the file to a remote server, we should have $PS_U = f$ and $PR_U = 0$. The remote server's send and receive amounts are are not included in these totals.

Using $H(r)$, the total bytes retrieved during the retrieval phase $DR_{H(r)}$ should be

$$DR_{H(r)} = \max(n - r, 0)f$$

Again, the total bytes sent $DS_{H(r)}$ should be equal to $DR_{H(r)}$ since no data leaves the network.

Using U, the total bytes retrieved during the retrieval phase DR_U should be $DR_U = (n-1)f$, and $DS_U = 0$.

When HDFS is used with a replication factor of r, the total bytes sent and received should not depend on r. It should only depend on f and n. If this is not the case, then the implementation is not distributing the data optimally. Note that

$$\begin{aligned} PS_{H(r)} + DS_{H(r)} &= (\min(r, n) - 1)f + (\max(n - r, 0))f \\ &= (\min(r, n) - 1 + \max(n - r, 0))f \end{aligned}$$

When $n > r$,

$$(\min(r, n) - 1 + \max(n - r, 0))f = n - r + r - 1 = (n - 1)f$$

When $n \leq r$,

$$f(\min(r, n) - 1 + \max(n - r, 0)) = (n - 1)f$$

Therefore $PS_{H(r)} + DS_{H(r)} = (n-1)f$, which does not depend on r.

For U, $PS_U + DR_U = f + f(n-1) = fn$.

Adding up all bytes sent and received (all wirelessly) by devices in the network during a given experiment, we get

$$PS_U + PR_U + DS_U + DR_U = nf$$

and

$$PS_{H(r)} + DS_{H(r)} + PR_{H(r)} + DR_{H(r)} = 2PS_{H(r)} + 2DR_{H(r)} = 2(n-1)f$$

This suggests that the total latency of U will be slightly more than half that of $H(r)$ for $n > 1$, assuming that a very fast connection exists between the router and the server and that the latency is determined primarily by the number of wireless transfers.

6.6.4 Results

Figure 6.19 shows the publishing time vs. f. Publishing time increases with f and with r since more time is required to send larger blocks and create additional replicas.

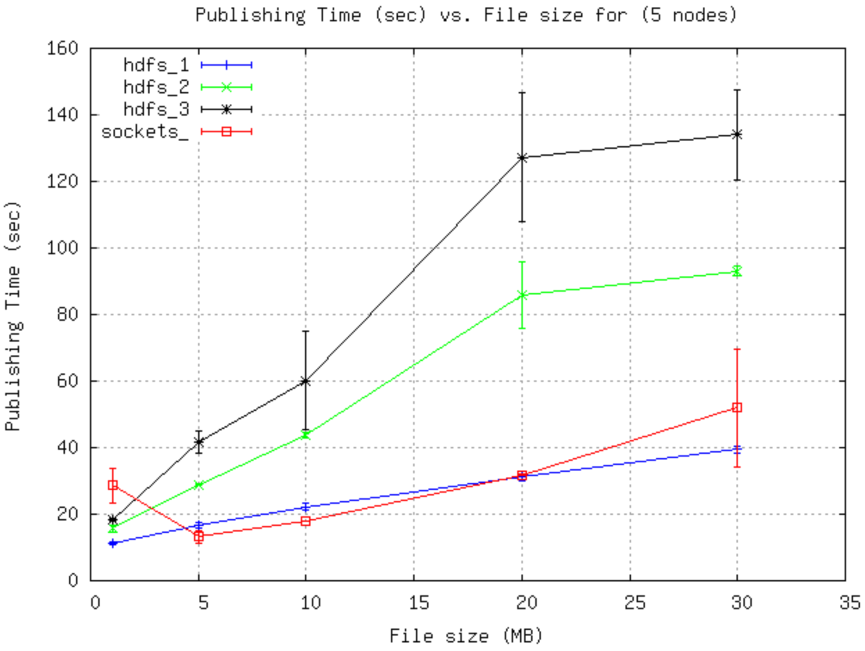


Figure 6.19: Publishing time vs. input size for 5 nodes.

Figure 6.20 shows the distribution of retrieval times vs. n, and Figure 6.21 shows the mean retrieval time for the same distribution. When U is used, retrieval time increases approximately linearly with n because of contention on the connection to the server. Retrieval time also increases approximately linearly when $H(1)$ is used since the retrieved blocks must be copied from one node to all $n-1$. Retrieval time for $H(1)$ increases about twice as fast as retrieval time for U with n, supporting our hypothesis. The range of retrieval times for $H(2)$ and $H(3)$ is significantly larger than that of $H(1)$ because of the $r-1$ nodes that can access local replicas quickly ($r-1=0$ when $r=1$). For a given n, the mean and median retrieval time decrease with r. This agrees with our hypothesis.

Retrieval times are low and have little variance when $r=n$. In this case, all blocks are retrieved from the local disk. Retrieval time for $H(r)$ remains lower than retrieval time for U for higher values of r as n increases.

We are ultimately interested in the total time required to transfer data from one node

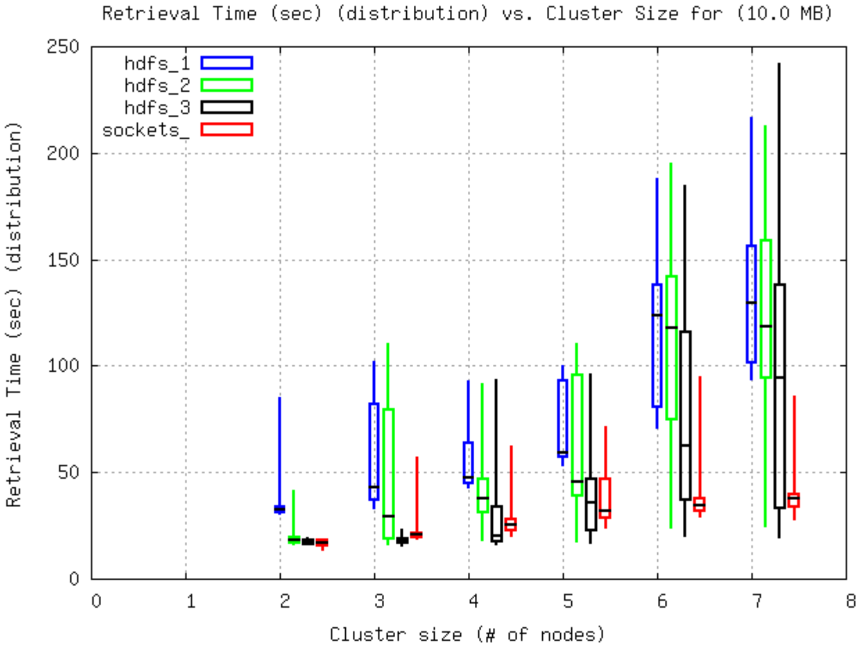


Figure 6.20: Retrieval time distribution vs. number of nodes for a 10 MB file. Points in box-and-whisker plot correspond to (from bottom to top) minimum, lower quartile, median, upper quartile, and maximum. Box-and-whisker plots are shifted slightly on the x-axis for visual clarity, but correspond to the nearest n to the left.

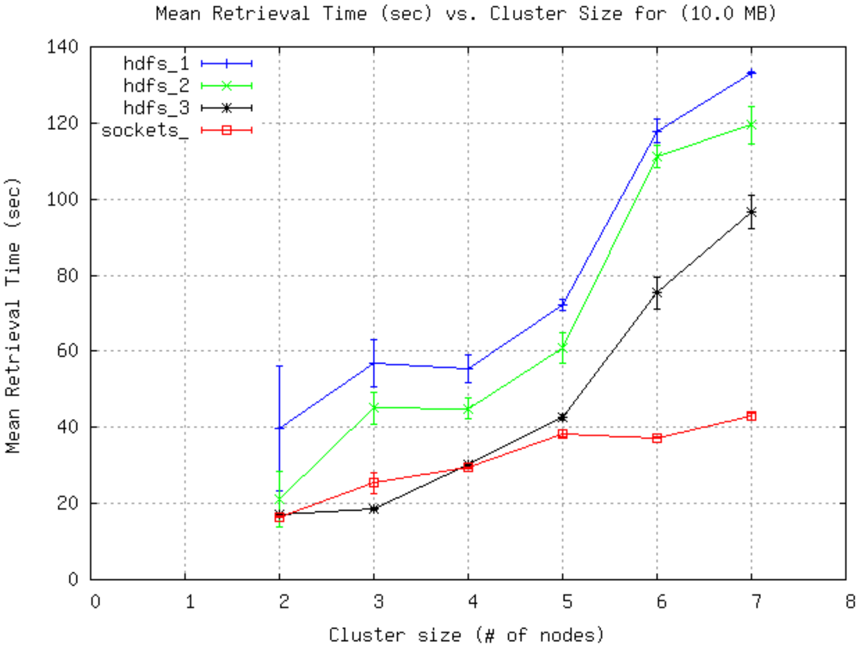


Figure 6.21: Mean retrieval time vs. number of nodes for a 10 MB file.

to $n-1$ other nodes, i.e. the total of the publishing and retrieval times. Figure 6.22 shows this total vs. n for cases U, $H(1)$, $H(2)$, and $H(3)$. U outperforms $H(r)$ significantly in all cases except when $n = 1$. This is expected because every byte is sent wirelessly twice in the case of $H(r)$, and only once in the case of U.

Figure 6.23 shows the total bytes sent or received by nodes in the cluster vs. n. Note that we count both the send and and the receive for a given transferred byte. This plot is exactly what our model predicts, i.e. $PS_U + DS_U + PR_U + DR_U = nf$ and $PS_{H(r)} + DS_{H(r)} + PR_{H(r)} + DR_{H(r)} = 2(n-1)f$, where $f = 20\text{MB}$ in this case.

Note that data collection on 8 and 9 node clusters failed because of hardware problems in the testbed. Some data was collected for these cases, but various failures contributed significantly to publishing and retrieval durations and thus we do not consider the results to be statistically valid.

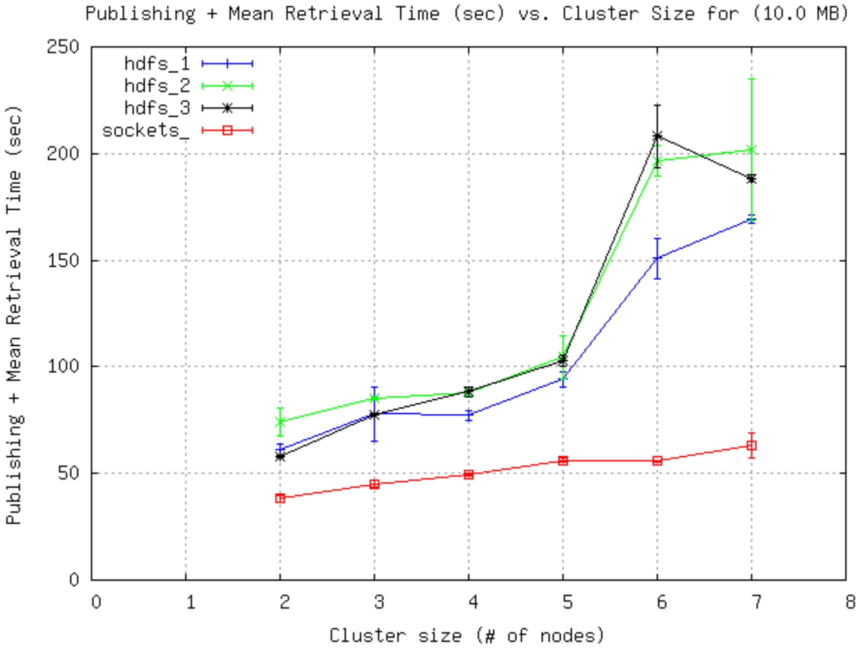


Figure 6.22: Total upload and mean download time vs. number of nodes for 10 MB file.

6.6.5 Conclusions

Although $H(r)$ allows for more parallelism in block serving for $r = 2$ and $r = 3$, the extra wireless transfers required to send data between two devices in the network compared to sending data to or receiving data from a server through the router (requiring only one wireless transfer in each case) prevent data from being published and served faster than in U. However, given that block serving times decreased with r, using $H(r)$ might yield better performance than U for sufficiently high n and r. Tests with higher n and r are warranted.

Note that a fast connection to this server was used in this experiment. In reality, the quality of connections to remote servers can vary, whereas local networks ensure high connection quality among nodes connected on the local network. Therefore Hyrax may still be preferable for data sharing in cases where a stable, high-bandwidth connection to a remote server is not guaranteed.

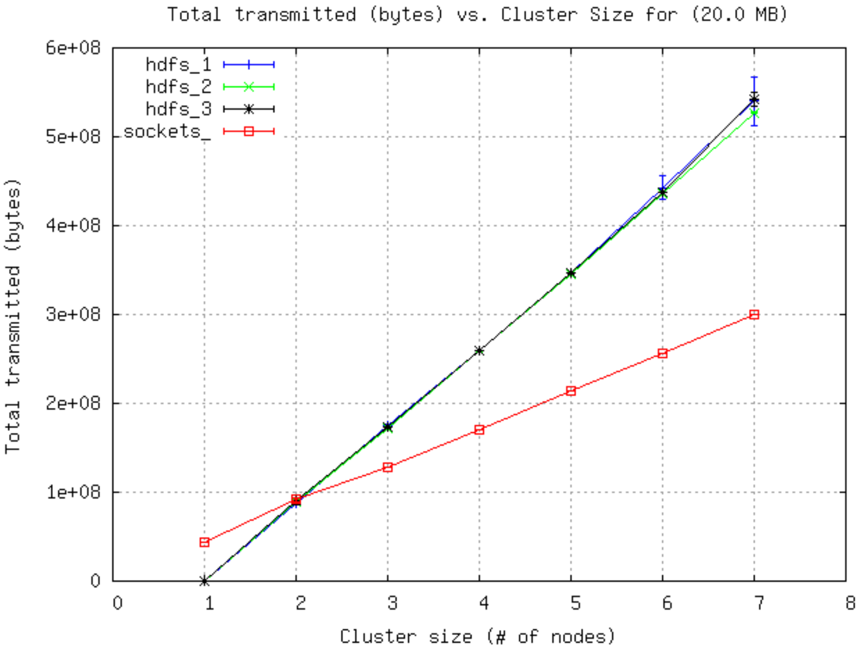


Figure 6.23: Total bytes sent or received vs. number of nodes for 20 MB file.

6.7 Battery consumption

Battery consumption is an important consideration in any mobile system. In this experiment, the rate at which battery is consumed by Hyrax is compared to that of other common tasks performed on mobile devices.

6.7.1 Question

In this experiment, we ask: how does the power consumption of Hyrax compare to that of other tasks?

6.7.2 Approach

We run several tasks on the phones and record battery levels over time. These tasks are:

1. **Video streaming from phone**. Qik Qik, Inc. is an Android application that streams

live video from the camera to the Internet. In this task, video is streamed to the Internet using Qik.

2. **Downloading**. Data is continuously downloaded from a server in the local network to the phone.

3. **Video recording**. Video is recorded using the Android video recording application.

4. **Hyrax Sort**. The Hyrax Sort benchmark is run repeatedly on a cluster of 3, 5 and 7 phones.

5. **Idle**. Other than the Android system services, no tasks are run on the phone.

Battery consumption rate R of a given run is estimated by using linear regression to fit a line to the the points of the battery level vs. time plots. The slope $\bar{R}$ and the correlation coeffecient of fitted line are reported for each task. Each workload is run on three different phones.

System resource usage logs are collected for each workload. The average network, disk, and CPU load (normalized by time, when applicable) is given for each workload in order to give more context to the battery life results. For the Hyrax workloads, Hadoop logs are also collected. Task times are extracted from these logs in order to correlate tasks with battery consumption. As in the performance experiment, the input size of the Sort workload is scaled with the number of nodes.

6.7.3 Hypothesis

Network transfers accounts for a large part of energy usage on smartphones. By design, Hadoop uses network bandwidth sparingly. Therefore we expect Hyrax to use less power than a network-bound application, but more power than a multimedia recording application which does not use the network.

We expect the video streaming workload to use networking, CPU, and disk heavily. We expect the video recording workload to use CPU and disk heavily. We expect the downloading workload to use networking heavily. We don't expect Hyrax to use any resource particularly heavily.

We expect reduce jobs to consume more battery both in total and per second than map jobs because they tend to take longer, and they use the network more. The "sort" and "shuffle" operations are considered parts of the reduce tasks. Furthermore, we expect battery consumption to increase with the number of nodes because, as we observed in the

Task	$\bar{R}$ (% / s)	r^2	$L_{\bar{R}}$
Video streaming	0.0151	0.981	1.8 hours
Video recording	0.0122	0.822	2.3 hours
Downloading	0.0102	0.939	2.7 hours
Hyrax Sort (3 nodes)	0.00479	0.992	5.8 hours
Hyrax Sort (5 nodes)	0.00537	0.935	5.2 hours
Hyrax Sort (7 nodes)	0.00580	0.969	4.8 hours
Idle	0.00008770	0.6453801	13.2 days

Table 6.8: Battery experiment results.

performance experiments, network sends and receives increase with the number of nodes (when the input is scaled with the number of nodes), and CPU utilization does not decrease significantly with the number of nodes for the Sort benchmark.

6.7.4 Results

Table 6.8 shows, for each task, the estimated battery consumption rate $\bar{R}$, the correlation coefficient for $\bar{R}$, and the total battery life of the G1 battery at the consumption rate of $\bar{R}$. We compute R in terms of battery % per second, but it can also be expressed in units of A or C/s. Recall from §6.1.1 that the capacity B of the G1 battery is 1150 mAh. For a battery with a capacity of B mAh, the conversion is:

$$R\%/\mathrm{s} = R\frac{B\mathrm{mAh}/100}{\mathrm{s}}\frac{3600\mathrm{s}}{\mathrm{h}} = 36RB\mathrm{mA}$$

Given that the consumption rate is $\bar{R}\%/\mathrm{s}$, the expected battery life $L_{\bar{R}}$ is

$$L_{\bar{R}} = \frac{100}{\bar{R}}\mathrm{s}$$

Battery life results are summarized in Figure 6.24. Figures 6.25 and 6.26 show battery level vs. time for video streaming and Hyrax Sort respectively.

Table 6.9 shows the resource usage of each workload. CPU utilization is reported as the mean over all readings. All other resources are reported in terms of mean count per second. The battery consumption of other hardware such as the camera and the screen is not accounted for by these statistics.

Table 6.10 and Figure 6.28 show the battery consumption of different task types in terms of battery % per second of task type and battery % per task. Figure 6.27 shows the

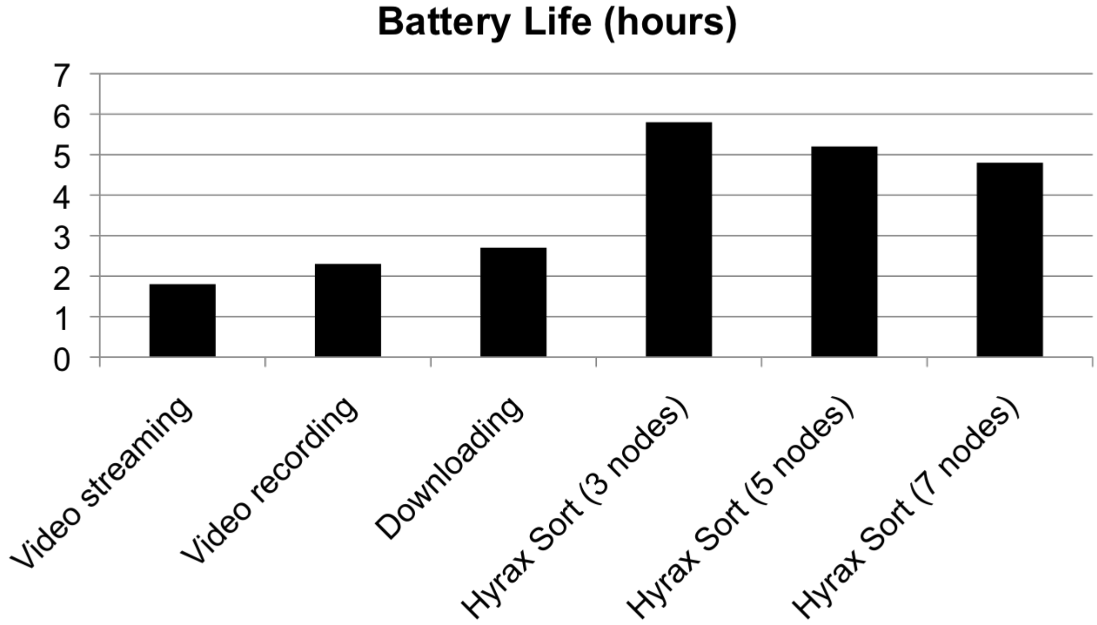


Figure 6.24: Battery life by task.

Resource	Video Streaming	Video Recording	Downloading	Hyrax Sort (5 nodes)
CPU	93.6 %	51.8 %	74.6 %	43.6 %
Disk reads	0.0119 reads/s	0.0288 reads/s	0.000 reads/s	0.0363 reads/s
Disk writes	0.589 write/s	0.473 writes/s	0.101 writes/s	0.802 write/s
Network send	34.7 KB/s	0.00196 KB/s	7.31 KB/s	2.00 KB/s
Network receive	0.811 KB/s	0.00163 KB/s	315 KB/s	1.84 KB/s

Table 6.9: Mean resource usage for each battery workload. Computed over entire duration of each workload and averaged over all phones.

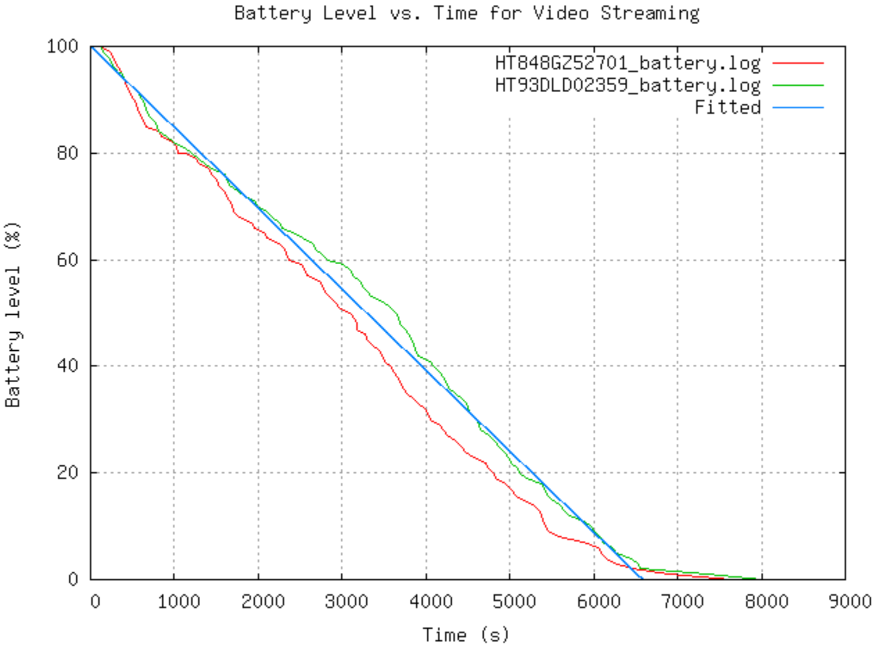


Figure 6.25: Battery consumption fit for video streaming battery level data.

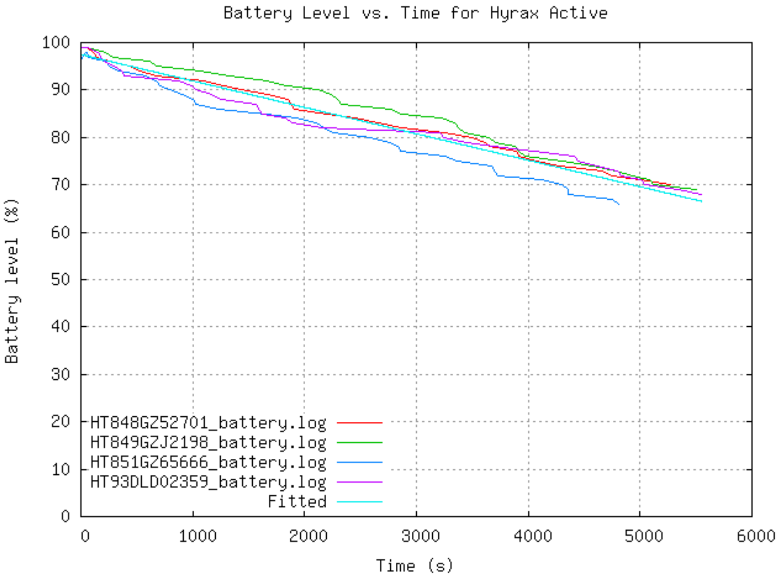


Figure 6.26: Battery consumption fit for Hyrax-active battery level data.

Task type	Battery % / second	Battery % / task
Map (successful)	0.00620	0.228
Map (failed)	0.00466	0.208
Reduce (successful)	0.00668	1.07
Reduce (failed)	0.00585	3.21

Table 6.10: Battery consumption by task type in Hyrax Sort (7 nodes).

breakdown of battery consumption of each task compared to the breakdown of total time spent in each task.

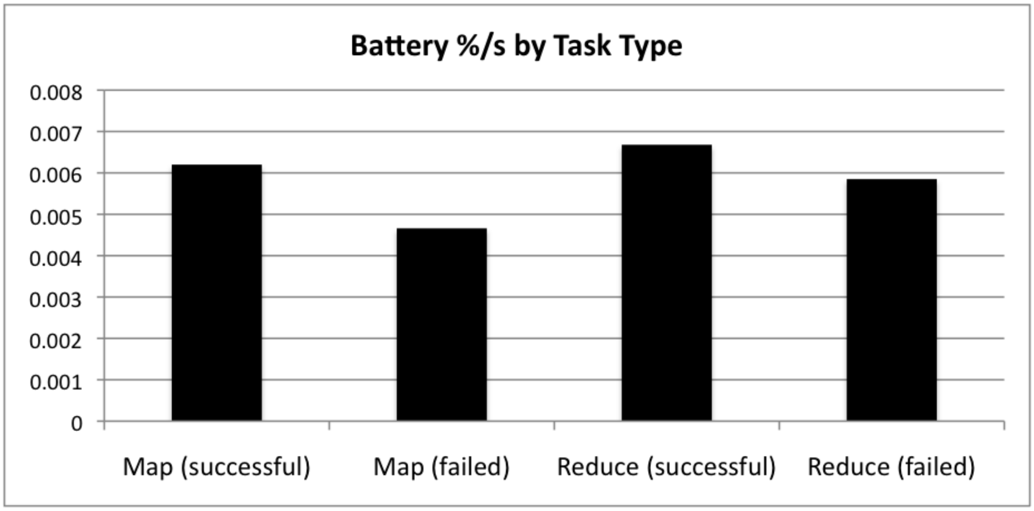


Figure 6.27: Battery consumption rates by task type for Hyrax Sort with 7 nodes.

6.7.5 Conclusions

Figure 6.24 shows that Hyrax, when running an intensive workload, consumes battery life at about half the rate of continuous downloading or video recording and slightly more than a third of the rate of video streaming. Unexpectedly, the video recording workload used much more battery than Hyrax, probably because of the power used by the camera and the screen. Given that this worst case power consumption rate for Hyrax is so much less than that of downloading, video recording, and video streaming, it seems reasonable. Furthermore, the implementation Hyrax has not been optimized for power at all, so there is probably a significant opportunity to improve its battery consumption.

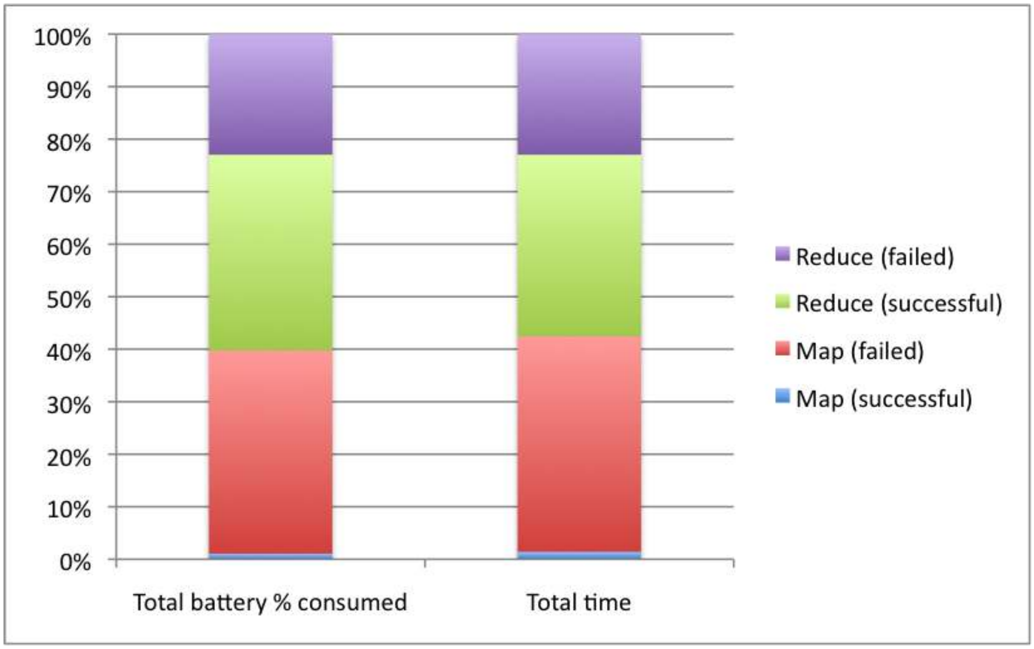


Figure 6.28: Normalized battery consumption and total time by task type for Hyrax Sort with 7 nodes.

Figure 6.24 also shows that battery consumption increases from 3 to 7 nodes. This is probably because of the additional network transfers that the reduce task must perform to collect the intermediate values for larger numbers of nodes.

Figure 6.27 suggests that battery consumption depends primarily on the amount of time spent in the task, not the task type. Differences in the power consumption of each task per second, shown in Figure 6.28, are not significant enough to identify parts of Hadoop that should be targeted to improve energy efficiency. More specific characterization of the MapReduce job would be required to make a specific battery consumption diagnosis.

Chapter 7

Case Study: Distributed Video Search and Sharing

To determine the advantages and drawbacks of Hyrax, an application was developed on it. A simplified version of the distributed mobile multimedia search and sharing application outlined in §1 was implemented and evaluated. This application would be useful at events where many mobile users want to record and share multimedia files.

The Hyrax multimedia search and sharing application, *HyraxTube*, allows users to browse through videos and images stored on a network of phones and search by time, location, and quality. Quality ratings based on sensor data are generated by periodically executing a MapReduce job. Requests are serviced by reading results generated by this MapReduce job from HDFS. The client interface is implemented as a web application so that it can be used on both mobile devices and desktop machines.

7.1 Requirements

The following requirements were established for HyraxTube:

1. Provide an interface for browsing files and searching by time, quality, and location.

2. Provide low-latency access to information about the files through a web interface.

3. Allow users to download any video or photo from the smartphones.

In addition to these application-level requirements, HyraxTube should scale with the number of devices and the number of users. It must also store data reliably. These properties are provided by Hyrax.

7.2 Design

In order to allow users to browse files, a list command on HDFS can be exected. This is a fast, cheap operation since it only involves communicating with the NameNode.

Search by time, quality, and location involves retrieving all files that match the input time range, quality range, or radius of the given location. At first, we considered executing a MapReduce search job for every request, comparing the metadata for every multimedia file against a filter corresponding to the user input. We quickly realized that this would cause unacceptable request latencies and not scale with the number of users. Instead, we decided to run a daemon which periodically executes a MapReduce job which summarizes the metadata into a form that can be efficiently accessed and searched in the front-end web server. The summarization task generates a quality rating based on accelerometer readings corresponding to the device on which and the time range during which the video was recording. The summarized results are stored on HDFS.

File transfers are handled differently depending on whether the client is inside the mobile network (and thus can establish direct communication with each node) or outside of it. If the client is outside of the network, then the server opens an input stream from the file through the HDFS interface and streams the data to the requesting client, acting as a passthrough. If the client is within the network, then the data is transferred directly from the DataNodes hosting the blocks of the file to the client.

To improve the performance of block serving and MapReduce jobs and to decrease the likelihood of data loss on HDFS, a DataNode and a TaskTracker are run along with the web server. The technique outlined in §5.7.1 is applied, assigning phones to `/phone-rack` and the server to `/server-rack`. Since there are only two racks, any file published from a phone whose replication factor is set to 3 will be replicated to the server. This makes it much less likely for the data to be lost when the original phone leaves the network and makes serving files to clients outside of the network much faster and taxing of mobile resources.

7.3 Implementation

HyraxType was fairly easy to implement and would have been much more difficult without using the cloud interface provided by Hadoop. The application has no knowledge of the physical details of the service that it requests files and executes compute jobs on.

The server application was implemented using the Ruby WEBrick library, JRuby, and the Hadoop libraries. WEBrick was used to serve web pages and handle user input. Using JRuby made it possible to use Hadoop's Java libraries.

The server runs independently of the Hyrax cluster and only interacts with it through Hadoop's HDFS and MapReduce interfaces. In other words, the application's frontend is totally decoupled from the distributed nature of its backend. This makes it easier to develop and maintain the application.

We faced two limitations of Hyrax. One is that accessing sensor readings in the time range of a video is very slow. In a map task, it is necessary to scan through the file until the target start time. Another obstacle was the memory limitations of Android. We were unable to implement a thumbnailing MapReduce job because it required too much memory to load an image.

7.4 Field testing at Mellon Arena

7.4.1 Background and Motivation

Sports teams have been making efforts to promote game attendance among young fans Viera [2008]. One approach has been to provide an interactive experience via mobile technology such as YinzCam Media [2009]. YinzCam allows game attendants to view replays from various angles and explore other relevant information using their mobile phones.

Another way to engage fans is to allow them to participate in game coverage using the cameras on their phones. These videos could be displayed occasionally on large displays in the area, incorporated into the television broadcast, and shared with other mobile users during the game. In order to sort through this data effectively, broadcasters and other mobile users would need to be able to search by recording time and location in the arena. An automatically generated quality rating would also be useful.

The trivial approach to implementing this is to distribute a mobile application similar to Qik that streams video to servers in the arena and provide another interface for downloading videos from other users, along with any other relevant information such as location and time of recording. There are several drawbacks to this approach. One is that this type of video streaming quickly drains battery, which was showed in §6.7. Another drawback is that this would require a substantial in-house hardware infrastructure. In order to guarantee reasonable performance at all times, enough servers and wireless networking resources would need to be provisioned to handle the maximum possible load from users uploading and downloading files. This would in many cases be prohibitively expensive.

Instead, Hyrax and (a more developed variant of) HyraxTube could be deployed and supported by a drastically smaller in-house hardware infrastructure. Using Hyrax, only videos of interest would ever be transferred over the network. Popular videos could be replicated to and served from an arbitrary fraction of phones in the arena. Node departure would be infrequent because spectators do not move around very much and don't leave the arena until towards the end of the game, allowing for a low default replication factor to be used. With today's mobile technology, a capable wireless network infrastructure would be required; however, within a few years, wireless ad hoc and mesh networking among phones will obviate such an infrastructure. A server would be needed to run a NameNode and a JobTracker for the Hyrax cluster, and any additional servers could be put to use for hosting block replicas and executing MapReduce tasks faster and without draining batteries of phones in the network, as described in §5.7.1.

7.4.2 Experiences at Mellon Arena

Hyrax was tested at Mellon Arena, the home arena of the Pittsburgh Penguins, using the wireless network infrastructure originally installed for YinzCam Media [2009]. This wireless network is implemented using Xirrus WiFi arrays Xirrus connected to several switches. Servers are connected behind these switches to allow fans to access game footage.

The first obstacle that we encountered was that the network had been configured to disable peer-to-peer networking. This setting was enabled for YinzCam to improve network performance. With this setting disabled, devices on the network were able to determine each other's MAC address, but not connect via TCP.

This experience showed that it may be non-trivial to integrate a peer-to-peer system into an existing wireless network that has been configured for access to remote services only. In the near future, we plan to investigate the issues that prevented peer-to-peer connections in Mellon Arena.

Chapter 8

Related Work

Our work on mobile-cloud computing is primarily related to previous work in mobile grid computing and mobile distributed filesystems. Hyrax is distinguished from all of the projects in these two fields because it combines distributed storage and distributed computation and provides a cloud interface to these capabilities that abstracts away from dealing with individual devices.

8.1 Mobile Grid Computing

Work has been done on systems that share resources and collaborate on computational tasks in mobile device networks. This has mostly been in the form of grid computing, which is an important part of cloud computing Myerson.

Litke et al. [2004] defines the "Grid" as "a distributed, high performance computing and data handling infrastructure that incorporates geographically and organizationally dispersed, heterogeneous resources (computing systems, storage systems, instruments and other real-time data sources, human collaborators, communication systems) and provides common interfaces for all these resources, using standard, open, general-purpose protocols and interfaces". Furthermore, the "Mobile Grid" is "full inheritor of the Grid with the additional feature of supporting mobile users and resources in a seamless, transparent, secure and efficient way." This fits well with our purpose in creating Hyrax: to provide a convenient abstraction and runtime system for utilizing the resources of a network of smartphones.

McKnight et al. [2004] gives an overview of the field of wireless grid computing. It discusses the additional capabilities offered by wireless grids and the new challenges faced by wireless grids compared to traditional grids. It also gives five requirements for wireless grid middleware: resource description, resource discovery, coordination, trust establishment, and clearing. In Hyrax, resources are described and provided via the HDFS interface. Coordination of data is performed by the NameNode, and coordination of computation is done by the JobTracker. Hyrax currently assumes trust, but this assumption may be removed by adding security and storage fault-tolerance in the future.

Ahuja and Myers [2006] provides a survey of wireless grid computing, following a structure similar to McKnight et al. [2004]. It points out the problem of frequent node connects and disconnects in mobile grids. Hyrax addresses this problem to some extent by relying on Hadoop's mechanisms for handling faulty nodes.

Mobile OSGI.NET Chu and Humphrey [2004] extends OSGI.NET, a grid computing implementation, to mobile devices. The goals of Mobile OSGI.NET are to provide better potential for collaboration among mobile devices, support collaboration among mobile devices with traditional, non-mobile computers, operate on many device platforms, and address the particular characteristics of mobile devices, including intermittent network connectivity and resource constraints. Mobile OSGI.NET and OSGI.NET on desktop machines are compared in terms of latency for basic operations and jobs, varying the number of devices and the workload size. Battery usage with varying workload sizes and number of devices is also presented. Hyrax is analogous to Mobile OSGI.NET in that it extends Hadoop to mobile devices while preserving interoperability with Hadoop on static machines, and we perform a quantitative comparison of Hyrax and Hadoop. However, our experiments go into significantly more depth than those of Mobile OSGI.NET, featuring more benchmarks, more devices, more resource usage statistics, more samples, and more investigation of the distributions of latencies. Unlike Mobile OSGI.NET, a demonstration application is developed on Hyrax.

Ibis for mobility Palmer et al. [2009] applies grid computing techniques to distributed computing on mobile devices, which includes integrating mobile phones into the grid. This included porting the Ibis grid computing platform to run on Android. Ibis also discusses the challenges of mobile distributed computing and presents a strong argument for distributed computing on mobile devices based on the growth in the Smartphone market and the pitfalls of cloud computing using proprietary services. Our work is structured in a similar way to Ibis for Mobility in that it involves porting an existing distributed system to run on a mobile platform, relying on the existing system's solutions to analogous problems between static and mobile grids. One drawback of Ibis is that Android emulators were used instead of physical Android devices, and no experimental evaluation was conducted. In contrast, Hyrax has been implemented, demonstrated, and experimentally evaluated on real Android phones.

WIPdroid Chou and Li [2008] is another distributed computing platform for Android. It is based on the Web Services Session Initiation Protocol (WIP), which allows "real-time service-oriented communication over IP". Using WIP, WIPdroid can provide a two-way web service interface similar to that of an online service supported by a "cloud" backend of mobile devices. Like Ibis, WIPdroid is developed and tested on Android emulators.

GridGain Systems has succeeded in running the GridGain cloud computing platform on Android phones Kharif [2008], but this is still in early stages of development. The GridGain architecture is probably the closest to Hadoop's of all of the grid systems that are being targeted at mobiles. GridGain directly supports deployment on a cloud, and MapReduce is an important feature of the system. Future work on Mobile GridGain could be directly compared to our work on Hyrax.

xSchapome of our motivational applications are inspired by mobile grid applications, which use the sensors and multimedia capture devices of a collection of mobile devices. Reades et al. [2007] monitors the locations of mobile users in an urban environment and studies the dynamics of mobile usage and crowd movement over time. Hull et al. [2006] uses mobile sensors for traffic analysis, and Lo et al. [2008] uses mobile device sensors for a similar task. McKnight et al. [2004] describes a distributed audio recording application using microphones from a mobile phone grid.

8.2 Sensor In-network Processing

Another approach to implementing cloud computing on mobile devices is to start with a wireless sensor network API and implementation. These systems are generally targeted at resource-limited embedded devices, and are therefore very good at preserving resources and handling faults that arise in wireless networks. These systems use in-network processing, i.e. summarization/computing on local nodes, to minimize data transfers Intanagonwiwat et al. [2003]. They provide high-level database interfaces for executing queries on distributed data Yao and Gehrke [2002], Bonnet et al. [2001], Madden et al. [2005], Deshpande et al. [2003]. Security support for wireless sensor network in-network processing has been studied Deng et al. [2003]. Efficient information sharing in wireless sensor networks has been studied in great depth Intanagonwiwat et al. [2003], W. R. Heinzelman and Balakrishnan [1999]. Sensor network architectures have also been developed for more powerful, resource-unconstrained multimedia sensors Campbell et al. [2005] and for networks with nodes of heterogeneous performance capabilities Tsiatsis et al. [2005]. Some of the applications of mobile-cloud computing, such as distributed image search, have been studied in a sensor network context Yan et al. [2008]. Considering all of these compatibilities, it would be worthwhile to investigate the usage of sensor network software for mobile-cloud computing in future work.

Nevertheless, sensor network software platforms have several limitations relative to a server-targeted distributed system such as Hadoop with respect to mobile-cloud computing. They are designed for collecting data and servicing queries from entities outside the network, typically through a special “gateway” node Suba et al. [2008]. In a mobile-cloud computing setting, clients would often run on nodes within the device cluster. Using a sensor network framework would require (without non-trivial modification) heavy data transfers through the gateway from “sensor” devices to “client” devices, whereas a distributed filesystem such as HDFS allows for peer-to-peer bulk transfers and direct access to local replicas when they are available.

The computations performed within sensor networks are targeted at efficient data collection and querying, not generic compute jobs. Although Hyrax is not intended to be used for generic distributed computing, it does provide much more flexibility in specifying computations. By distributing executable code, MapReduce jobs on Hyrax can process sensor, multimedia, text, and other data in arbitrary ways. The sensor network database concept is well-suited for applications such as sensor maps and traffic monitoring (described in §3.3.2), but it would not work well for non-sensor applications such as multimedia search, multimedia sharing, and social networking, where MapReduce jobs would be used to process text and multimedia data.

In sensor networks, raw data is processed purely locally, not on other nodes. This is ideal for preserving power by avoiding network transfers, but it limits the dynamic adaptablity of job execution. Although MapReduce prefers to process data locally, it is capable of offloading computation to other nodes when necessary. The filesystem interface enables transparent access to data on other nodes, allowing for both in-network data offloading and applications built around data-sharing.

Using a server-targeted platform such as MapReduce offers trivial compatibility and cooperation between devices running the ported application and servers running the application. Hyrax could easily plug into an existing Hadoop cluster without modifying the Hadoop cluster code or configuration. Starting with a sensor network platform would require porting to both mobile devices and servers, which would be much less convenient and probably lead to more divergence in compatibility.

In some sense, Hyrax links sensor network systems with large-scale data-intensive computing platforms for servers by showing how and to what extent solutions for fault-tolerance on server networks (replication, re-execution, speculative execution, etc.) can be applied to fault-tolerance mobile device networks. Running Hadoop on a mobile platform and (hypothetically) running a sensor network platform on servers illustrate what design aspects can be shared between the two environments and which aspects require different solutions.

8.3 Mobile Data Sharing Systems

One major aspect of Hyrax that distinguishes it from mobile grid computing platforms is its use of a distributed filesystem, HDFS, for sharing data. Separate work has been done on distributed filesystems, peer-to-peer file-sharing, and other forms of data sharing on mobile devices. In contrast to most of the mobile-targeted distributed filesystems discussed in this section, HDFS is designed to handle large, unchanging files. This limitation is acceptable for the type of data that it is useful to store and share among mobile devices.

Coda Kistler and Satyanarayanan [1992], Satyanarayanan et al. [1993], Satyanarayanan [1996a], Mummert et al. [1995] was the first distributed filesystem to be investigated on a mobile platform. Coda inherits much of the design and functionality of AFS Howard et al. [1988]. It is used by clients as a location-transparent global UNIX filesystem. The file namespace is mapped to individual file servers. Coda supports disconnected operation, allowing clients to access and modify files even when disconnected from the network. Disconnected operation can also be used to save power by avoiding network transfers. Coda is used in conjunction with Venus, a client-side cache that is responsible for hoarding data, emulating operations on this data, and resolving changes in the data upon reconnecting to the network. Optimizations for operation in the presence of weak connectivity have also been integrated into Coda.

In Hyrax, files are accessed through a similar global interface which maps file paths to data stored on nodes in the cluster. Hyrax does not allow for disconnected operation because of its dependence on the NameNode for mapping file paths to data blocks. Furthermore, Hyrax discourages file modification once a file has been created. Without disconnected operation and file modification, the challenge of resolving file change conflicts is moot. Unlike Coda, Hyrax does not depend on a central set of servers to host data. Instead, it uses mobile devices themselves as block servers with the option of adding static servers to improve reliability and performance.

Several other filesystems optimized for mobile constraints have been studied. LBFS Muthitacharoen et al. [2001] is a network filesystem for low-bandwidth networks such as wireless networks. It exploits commonalities of a file before and after changes to avoid sending the entire file when a small change is made. Boukerche and Al-Shaikh [2007] implements another DFS client that prevents conflicts. Virtual Memory based Mobile Distributed File System Bagchi [2007] implements another thin-client mobile DFS that ensures consistency. Like Coda, all of these mobile filesystems use the mobile device only as a client, not as a host. Thus, while they face similar constraints as Hyrax, they do not solve the same problems.

M-DFS Michalakis [2004] implements "ephemeral filesharing" among mobile devices using the NFS protocol. M-DFS establishes a temporary distributed filesystem that allows mobile devices to access files stored on other devices in the network. M-DFS is more closely related to Hyrax than the thin-client mobile network filesystems because it involves sharing directly between devices, using mobile devices as both clients and servers. However, Hyrax is not really intended to be used in such a transient way. Through replication, a Hyrax network can promote long term data availability.

Kelényi et al. [2007] explores peer-to-peer file sharing on mobile devices, including a discussion on implementations of Gnutella Clip2 and BitTorrent Cohen, Bram for the Symbian mobile platform. Various aspects of peer-to-peer file sharing on mobile networks are studied in Ding and Bhargava [2004], Marossy et al. [2004], Zhiyuan et al. [2007], Lindemann and Waldhorst [2002], Data et al. [2001], Hofeld et al. [2005], Kurt [extern] The architectures of peer-to-peer file sharing systems are similar to that of HDFS in that file transfers are executed directly between peers, data is only stored on client nodes, replicas of data exist on multiple nodes, and there are high-level interfaces for retrieving files. However, Hyrax provides a filesystem abstraction on top of its peer-to-peer nature, making it more suitable for developing large-scale applications that are oblivious to the underlying implementation of the storage system.

Mobile data sharing systems have also been used to reduce traffic on cellular data networks. The Cellular-based Ad hoc Peer Data Sharing system (CAPS) Lee et al. [2005] uses devices on the mobile network as caches for data from remote sources. A subset of the devices are used as directory services for cache lookups. Hyrax also supports the goal of reducing load in data networks by processing data in-place and allowing files to be served directly from devices in the local network.

Chapter 9

Conclusions

Cloud computing using mobile devices has many advantages over traditional cloud computing for applications that use mobile data. Hyrax provides an infrastructure for mobile-cloud computing, providing an abstract interface for using data and executing computing jobs on a mobile device cloud.

Hadoop provides most of the essential features for a mobile-cloud computing infrastructure, making it suitable to use as a basis for Hyrax. Futhermore, there are several solutions provided by Hadoop that can be directly applied to challenges in a mobile computing environment, such as using fault-tolerance for tolerating node departure.

Unfortunately, Hadoop is fairly heavy-weight for current smartphone platforms. This is demonstrated by the high overhead costs of running MapReduce jobs on phones in our performance experiments. This overhead cost is exacerbated by by artificial limitations created by Android, such as the 16 MB application memory limit. Nevertheless, Hyrax easily scales to all of the nodes in our testbed, and would likely scale to many more nodes. It also works reasonably well for local peer-to-peer data sharing and is generally successful in tolerating node-departure.

Our experiences in implementing the distributed multimedia search and sharing application suggest that Hyrax provides a convenient, sufficiently abstract interface for developing applications that use mobile data.

9.1 Future work

Initial work on Hyrax creates many opportunities for enhancing the system and bringing it closer to real-world deployment.

9.1.1 NAT and firewall traversal

As noted in §4.1, Hyrax currently only works for sets of smartphones that can connect to each other via plain TCP/IP sockets. Because of this limitation, Hyrax cannot be used in a realistic setting where smartphones don't all have global, unrestricted IP addresses or aren't connected to the same local network. Real-world peer-to-peer applications use overlay protocols such as SIP Rosenberg et al. [2001] and JXTA Sun Microsystems [a] to get around NAT and firewall issues. SmartSockets were used in the Ibis mobile grid-computing project Palmer et al. [2009] to address this problem.

In a large, evolving codebase such as Hadoop's, it is wise to avoid changing code whenever possible. Instead, it is better to replace the underlying implementations of high-level interfaces. In the case of sockets, Hadoop creates sockets using the abstract SocketFactory class, whose implementation can be specified in Hadoop's configuration.

JXTA includes a peer-to-peer SocketFactory implementation. Incompatibilities between the Dalvik VM and the JXTA library have prevented us from using JXTA sockets within Hadoop, but getting this to work is just a matter of investing the time to find and work around this incompatibility.

9.1.2 Battery consumption analysis and improvement

The battery results presented in this paper only scratch the surface of understanding the power consumption of Hyrax and thus Hadoop. As noted in §4.3.1, there is interest among power-users of Hadoop in improving its energy efficiency in order to reduce environmental impact and energy costs. Insights into how Hadoop consumes battery on a mobile platform may be applicable to improving Hadoop's power consumption in a server cluster.

In particular, more experiments should be performed to determine the contributions of different tasks of different jobs to battery consumption for varying numbers of nodes and input sizes. Such experiments were performed for MapReduce in a server setting in Chen et al. [2009]. Battery data could not be collected during our performance experiments because of limitations in our testbed (namely, phones needed to be plugged in in order to be controlled via ADB, and there is no option to disable charging). Correlating battery consumption with task execution details may yield insights into what parts of Hadoop should be targeted for increasing energy efficiency.

9.1.3 Handling network changes

One important challenge that remains to be addressed is coping with network connection changes. In a situation where the central Hadoop services are running on a globally-accessible machine (as opposed to a machine inside a local network), a smartphone should be able to connect to the central machine as it changes networks. The central services would need to be able to identify the node regardless of its IP. Hadoop currently identifies nodes by their hostname. This issue could be addressed in Hyrax by using unique IDs separate from hostnames to identify DataNode and TaskTracker instances. Hyrax would also have to re-assign the "rack" of the device depending on which network it is on and attempt to rebalance block replicas according to the new topology.

9.1.4 Cluster selection

Another useful Hyrax feature would be plugging into different Hadoop clusters. At the application or configuration level, the user would be presented with a choice among several reachable clusters. Clusters might be set up for specific events or locations to support multimedia and sensor data gathering and processing. To implement this, a function for switching NameNodes would would need to be added to the DataNode. The DataNode would need to use a different metadata directory for each cluster that it connects to.

9.1.5 Mobile rack-awareness

"Rack by network distance", i.e. assigning a "rack" to each node based on its distance from other nodes in the network, has not been implemented yet. Hadoop uses rack information to select pairs of nodes for block transfers and determine where to place block replicas. It assumes that nodes on the same rack can communicate more quickly and cheaply.

In the case of smartphones, racks are analogous to sets of devices on local networks. These local networks may be implemented, for instance, by a WiFi router, an ad hoc WiFi configuration, or a peer-to-peer mesh network. Local networks such as these tend to have lower latencies and higher bandwidth and require less power to transmit data compared to connections to mobile data networks. Therefore it makes sense for Hadoop to treat locally-networked mobile devices into "racks". Implementing rack assignment for mobile devices could be implemented by matching up nodes based on their global IP addresses or by empirically determining the latencies between them by executing small transfers.

9.1.6 Sensor databases

Sensor logs in Hyrax are currently stored per-phone in flat text files. This makes it difficult to use the sensor data usefully. With a querying interface like SQL and a database infrastructure to support it, it would be easier and more efficient to perform operations such as range queries and joins to associate different sensor readings among different devices with each other. Existing database systems based on Hadoop such as HBase or Hive could be used to implement this.

9.1.7 Adaptive replication

It is very important to control the replication factors r in Hyrax. As pointed out in §5.5, the replication factor must be set to balance battery usage and data availability. A simplistic way of doing this is to adjust r_f depending on how many times f is requested. However, there may be more effective ways to adapt r_f to suit the access patterns of a particular cluster. r_f should be increased when f is popular to increase parallelism in block serving, but it should decrease when the f is not as popular to save disk space. A technique for adapting r_f to balance availability, battery usage, and disk usage should be developed

9.1.8 Security

Hyrax stores data on many devices, each with a different owner. There is nothing preventing owners of devices on which blocks are stored from reading the contents of data blocks. In order to prevent device owners from reading the contents of files that they don't have permission to read, data blocks can be encrypted such that only those users who have permission to the corresponding files can read their contents.

This can be implemented using a public key encryption scheme. Creators of the file would encrypt each file using a randomly generated key. This key would then be encrypted using the public key of each user who has access to the file. A central table of (user, filename, encrypted key) triples would be stored and accessed using some secure authentication system. After retrieving their encrypted key for a given file, a user would decrypt the key using their (locally stored) private key, and then use the resulting key to decode the blocks of the file. Using this encryption scheme, MapReduce would have to be modified to run tasks for data from a given input file only on nodes that can decrypt the file.

A scheme for encryption in network-attached storage systems (which are similar to HDFS) is developed in Miller et al. [2002].

9.1.9 Storage fault-tolerance

Device owners must be assumed have unlimited control over their systems, including the data that is stored on them. Thus it cannot be assumed that the output of any node, including block data and intermediate values in MapReduce computations, is valid.

The problem of fault-tolerance in HDFS is reducible to the Byzantine Generals Problem Lamport et al. [1982]. Techniques for low-overhead Byzantine fault-tolerance in distributed storage systems were developed in Hendricks et al. [2007]. These techniques could be applied directly to implement fault-tolerant storage in HDFS. Fault tolerance for MapReduce tasks could potentially be implemented by enabling speculative execution and voting on intermediate values.

9.1.10 Optimization or re-implementation of MapReduce

In Hyrax, MapReduce jobs are much slower for a given input size than they are on server clusters. This is partially caused by resource limitations, such as the extremely small amount of memory available for buffering intermediate values, and partially by the MapReduce implementation. It may be possible to optimize MapReduce to use resources more efficiently or to reimplement MapReduce in a simpler, more mobile-friendly way.

9.1.11 Large-scale testing

So far, Hyrax has only been proven to scale to the 12 phones in our testbed. It should not be assumed that this trend will apply to 100, 1000, 10000, or more phones. Testing on larger numbers of phones should be performed.

9.1.12 Offloaded vs. local computation

In this paper, the tradeoffs between local and offloaded computation have not been quantified. It would be useful to develop a model for determining when it is preferable to offload some job to remote servers and when to perform the job locally considering input size, network speeds, expected battery consumption, and system resource availability.

Bibliography

Sanjay P. Ahuja and Jack R. Myers. A survey on wireless grid computing. *J. Supercomput.*, 37(1):3–21, 2006. ISSN 0920-8542. doi: http://dx.doi.org/10.1007/s11227-006-3845-z.

Ian F. Akyildiz, Tommaso Melodia, and Kaushik R. Chowdhury. A survey on wireless multimedia sensor networks. *Computer Networks*, pages 921–960, 2006.

Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS '67 (Spring): Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, New York, NY, USA, 1967. ACM. doi: 10.1145/1465482.1465560. URL `http://dx.doi.org/10.1145/1465482.1465560`.

Android. Designing for performance. `http://bit.ly/17ojgU`, a.

Android. Android Developers: WifiManager.WifiLock. `http://bit.ly/WxZel`, b.

Apache. Hadoop wiki - PoweredBy. `http://wiki.apache.org/hadoop/PoweredBy`.

Apache. Hadoop. `http://hadoop.apache.org/core/`.

Apache. Apache Harmony. `http://harmony.apache.org/`.

Apache. HBase. `http://hadoop.apache.org/hbase/`, a.

Apache. Hive. `http://wiki.apache.org/hadoop/Hive`, b.

Atebits. Tweetie. `http://www.atebits.com/tweetie-iphone/`.

Susmit Bagchi. Vmdfs: The design architecture, model and paging latency. In *MUE '07: Proceedings of the 2007 International Conference on Multimedia and Ubiquitous*

Engineering, pages 1004–1009, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2777-9. doi: http://dx.doi.org/10.1109/MUE.2007.217.

Cory Beard. Preemptive and delay-based mechanisms to provide preference to emergency traffic. *Comput. Netw. ISDN Syst.*, 47(6):801–824, 2005. ISSN 0169-7552. doi: http://dx.doi.org/10.1016/j.comnet.2004.07.021.

Bluetooth SIG. Bluetooth technology gets faster with bluetooth 3.0. `http://bit.ly/kfnnB`, April 2009.

Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Towards sensor database systems. In *MDM '01: Proceedings of the Second International Conference on Mobile Data Management*, pages 3–14, London, UK, 2001. Springer-Verlag. ISBN 3-540-41454-1.

Dhruba Borthakur. *The Hadoop Distributed File System: Architecture and Design*. The Apache Software Foundation, 2007.

Azzedine Boukerche and Raed A. Al-Shaikh. Towards building a conflict-free mobile distributed file system: Research articles. *Concurr. Comput. : Pract. Exper.*, 19(8): 1237–1250, 2007. ISSN 1532-0626. doi: http://dx.doi.org/10.1002/cpe.v19:8.

Jason Campbell, Phillip B. Gibbons, Suman Nath, Padmanabhan Pillai, Srinivasan Seshan, and Rahul Sukthankar. Irisnet: an internet-scale architecture for multimedia sensors. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 81–88, New York, NY, USA, 2005. ACM. ISBN 1-59593-044-2. doi: http://doi.acm.org/10.1145/1101149.1101162.

Yanpei Chen, Laura Keys, and Randy H. Katz. Towards energy efficient mapreduce. Technical Report UCB/EECS-2009-109, EECS Department, University of California, Berkeley, Aug 2009. URL `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-109.html`.

Wu Chou and Li Li. WIPdroid - a two-way web services and real-time communication enabled mobile computing platform for distributed services computing. In *SCC '08: Proceedings of the 2008 IEEE International Conference on Services Computing*, pages 205–212, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3283-7-02. doi: http://dx.doi.org/10.1109/SCC.2008.113.

David C. Chu and Marty Humphrey. Mobile OGSI.NET: Grid computing on mobile devices. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on*

Grid Computing, pages 182–191, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2256-4. doi: http://dx.doi.org/10.1109/GRID.2004.44.

Clip2. The Gnutella Protocol Specification v0.4. `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf`.

Cohen, Bram. The BitTorrent Protocol Specification. `http://www.bittorrent.org/beps/bep_0003.html`.

Corsair. USB Flash Wear-Leveling and Life Span. `http://bit.ly/BtKB3`.

Cooperation On Data, Maria Papadopouli, and Henning Schulzrinne. Effects of power conservation, wireless coverage and. In *In Proc. IEEE MobiHoc 01*, pages 117–127, 2001.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/1327452.1327492.

Jing Deng, Richard Han, and Shivakant Mishra. Security support for in-network processing in wireless sensor networks. In *SASN '03: Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 83–93, New York, NY, USA, 2003. ACM. ISBN 1-58113-783-4. doi: http://doi.acm.org/10.1145/986858.986870.

Denys Vlasenko. BusyBox. `http://www.busybox.net/`.

Amol Deshpande, Suman Nath, Phillip B. Gibbons, and Srinivasan Seshan. Cache-and-query for wide area sensor databases. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 503–514, New York, NY, USA, 2003. ACM. ISBN 1-58113-634-X. doi: http://doi.acm.org/10.1145/872757.872818.

Gang Ding and Bharat Bhargava. Peer-to-peer file-sharing over mobile ad hoc networks. *Pervasive Computing and Communications Workshops, IEEE International Conference on*, 0:104, 2004. doi: http://doi.ieeecomputersociety.org/10.1109/PERCOMW.2004.1276914.

Deborah Estrin, David Culler, Kris Pister, and Gaurav Sukhatme. Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1(1):59–69, 2002. ISSN 1536-1268. doi: http://dx.doi.org/10.1109/MPRV.2002.993145.

Inc. Facebook. Facebook. `http://www.facebook.com`.

Christos Faloutsos. *Searching Multimedia Databases by Content*. Kluwer Academic Publishers, Norwell, MA, USA, 1996. ISBN 0792397770.

Google. Efficient Computing. `http://www.google.com/corporate/green/datacenters`.

Matt Hamblen. Smart phones lead market growth. `http://bit.ly/Pn2o9`.

James Hendricks, Gregory R. Ganger, and Michael K. Reiter. Low-overhead byzantine fault-tolerant storage. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 73–86, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-591-5. doi: http://doi.acm.org/10.1145/1294261.1294269.

John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst.*, 6(1):51–81, February 1988. ISSN 0734-2071. doi: 10.1145/35037.35059. URL `http://dx.doi.org/10.1145/35037.35059`.

Sanjay Ghemawat Howard, Howard Gobioff, and Shun-tak Leung. The google file system, 2004.

Tobias Hofeld, Kurt Tutschku, Frank uwe Andersen, Hermann De Meer, and Jens O. Oberender. Simulative performance evaluation of a mobile peer-to-peer file-sharing system. In *In Next Generation Internet Networks NGI2005*, page 2005, 2005.

HTC. HTC Dream specification. `http://www.htc.com/www/product/dream/specification.html`, a.

HTC. HTC Magic specification. `http://www.htc.com/www/product/magic/specification.html`, b.

Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. Cartel: a distributed mobile sensor computing system. In *In 4th ACM SenSys*, pages 125–138, 2006.

IEEE. IEEE Computer Society LAN/MAN Standards Committee, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 4: Further Higher Data Rate Extension in the 2.4GHz Band. *IEEE Std 802.11g-2003*, 2003.

Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, 11(1):2–16, 2003. ISSN 1063-6692. doi: http://dx.doi.org/10.1109/TNET.2002.808417.

Intel. Data Center Efficiency. http://www.intel.com/technology/eep/data-center-efficiency/.

International Telecommunication Union. Measuring the Information Society - The ICT Development Index 2009. 2009. URL http://www.itu.int/ITU-D/ict/publications/idi/2009/material/IDI2009_w5.pdf.

Joe Farren. Wireless Industry Continues Efforts to Boost Networks in Preparation for Presidential Inauguration. http://bit.ly/xfgfi.

Imre Kelényi, Gergely Csúcs, Bertalan Forstner, and Hassan Charaf. Peer-to-peer file sharing for mobile devices. pages 311–324. 2007. doi: 10.1007/978-1-4020-5969-8_15. URL http://dx.doi.org/10.1007/978-1-4020-5969-8_15.

Olga Kharif. A warm welcome for android. *BusinessWeek*, January 2008.

James J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Trans. Comput. Syst.*, 10(1):3–25, 1992. ISSN 0734-2071. doi: http://doi.acm.org/10.1145/146941.146942.

Eric Knorr and Galen Gruman. What cloud computing really means. http://bit.ly/wX6V.

Tutschku Kurt(extern), T. Hofeld(extern), and F.-U. Andersen(extern). Mapping of file-sharing onto mobile environments: Feasibility and performance of edonkey with gprs. In *Proceedings of the WCNC 2005*, New Orleans, LA USA, 2005.

Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4:382–401, 1982.

Kang-Won Lee, Young-Bae Ko, and Thyaga Nandagopal. Load mitigation in cellular data networks by peer data sharing over wlan channels. *Computer Networks*, 47(2):271 – 286, 2005. ISSN 1389-1286. doi: DOI:10.1016/j.comnet.2004.07.009. URL http://www.sciencedirect.com/science/article/B6VRG-4D1MTXK-8/2/66a0bcbbf2bce9af183147f53d4db736. Wireless Internet.

William Lehr and Lee W. McKnight. Wireless internet access: 3g vs. wifi? *Telecommunications Policy*, 27(5-6):351–370, 2003. doi: 10.1016/S0308-5961(03)00004-1. URL `http://dx.doi.org/10.1016/S0308-5961(03)00004-1`.

Christoph Lindemann and Oliver P. Waldhorst. A distributed search service for peer-to-peer file sharing in mobile applications, 2002.

Antonios Litke, Dimitrios Skoutas, and Theodora Varvarigou. Mobile grid computing: Changes and challenges of resource management in a mobile grid environment. In *PAKM 2004 Conference*, 2004.

Chia-Hao Lo, Wen-Chih Peng, Chien-Wen Chen, Ting-Yu Lin, and Chun-Shuo Lin. Carweb: A traffic data collection platform. In *Mobile Data Management, 2008. MDM '08. 9th International Conference on*, pages 221–222, April 2008. doi: 10.1109/MDM.2008.26.

Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/1061318.1061322.

K. Marossy, G. Csucs, B. Bakos, L. Farkas, and J.K. Nurminen. Peer-to-peer content sharing in wireless networks. In *Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004. 15th IEEE International Symposium on*, volume 1, pages 109–114 Vol.1, Sept. 2004.

Matt. Flash Memory Trends. `http://www.mattscomputertrends.com/flashmemory.html`.

Lee W. McKnight, James Howison, and Scott Bradner. Guest editors' introduction: Wireless grids–distributed resource sharing by mobile, nomadic, and fixed devices. *IEEE Internet Computing*, 8(4):24–31, 2004. ISSN 1089-7801. doi: http://doi.ieeecomputersociety.org/10.1109/MIC.2004.14.

Yinz Media. Yinzcam. `http://www.yinzcam.com`, 2009.

Nikolaos Michalakis. Designing an nfs-based mobile distributed file system for ephemeral sharing. In *in Proximity Networks, Proc. of 4 th Workshop on Applications and Services in Wireless Networks, IEEE CS*. Press, 2004.

Ethan L. Miller, William E. Freeman, Darrell D. E. Long, and Benjamin C. Reed. Strong security for network-attached storage. In *USENIX Conference on File and Storage Technologies (FAST)*, pages 1–14, January 2002. URL citeseer.ist.psu.edu/miller02strong.html.

Lily B. Mummert, Maria R. Ebling, and M. Satyanarayanan. Exploiting weak connectivity for mobile file access. pages 143–155, 1995.

Athicha Muthitacharoen, Benjie Chen, David Mazieres, and David Mazi Eres. A low-bandwidth network file system. pages 174–187, 2001.

Judith Myerson. Cloud computing versus grid computing. http://bit.ly/16kRAk.

Occipital. Android performance 3: iphone comparison. http://occipital.com/blog/2008/11/02/android-performance-3-iphone-comparison/.

Open Handset Alliance. Android. http://www.android.com/.

Nicholas Palmer, Roelof Kemp, Thilo Kielmann, and Henri Bal. Ibis for mobility: solving challenges of mobile computing using grid techniques. In *HotMobile '09: Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, pages 1–6, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-283-2. doi: http://doi.acm.org/10.1145/1514411.1514426.

Will Park. Obama's Presidential Inauguration Ceremony wreaks havoc on wireless networks. http://bit.ly/Gc4ei, 2009.

Qik, Inc. Qik. http://qik.com/.

Jonathan Reades, Francesco Calabrese, Andres Sevtsuk, and Carlo Ratti. Cellular census: Explorations in urban data collection. *IEEE Pervasive Computing*, 6(3):30–38, 2007. ISSN 1536-1268. doi: 10.1109/MPRV.2007.53. URL http://dx.doi.org/10.1109/MPRV.2007.53.

Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnson, Jon Peterson, Robert Sparks, Mark Handley, and Eve Schooler. Sip: Session initiation protocol, 2001.

M. Satyanarayanan. Mobile information access, 1996a.

M. Satyanarayanan. Fundamental challenges in mobile computing. In *Symposium on Principles of Distributed Computing*, pages 1–7, 1996b. URL citeseer.ist.psu.edu/satyanarayanan96fundamental.html.

M. Satyanarayanan, James J. Kistler, Lily B. Mummert, Maria R. Ebling, Puneet Kumar, and Qi Lu. Experience with disconnected operation in a mobile computing environment. In *In Proceedings of the 1993 USENIX Symposium on Mobile and Location-Independent Computing*, pages 11–28, 1993.

Shazam Entertainment Ltd. Shazam. `http://www.shazam.com/`.

Smule. Ocarina. `http://ocarina.smule.com/`.

Alessandro Sorniotti, Laurent Gomez, Konrad Wrona, and Lorenzo Odorico. Secure and trusted in-network data processing in wireless sensor networks: a survey. *Journal of Information Assurance and Security*, 2:189–199, 2007.

Filip Suba, Christian Prehofer, and Jilles van Gurp. Towards a common sensor network api: Practical experiences. In *SAINT '08: Proceedings of the 2008 International Symposium on Applications and the Internet*, pages 185–188, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3297-4. doi: http://dx.doi.org/10.1109/SAINT.2008.60.

Sun Microsystems. JXTA. `https://jxta.dev.java.net/`, a.

Sun Microsystems. SunSPOTs. `http://www.sunspotworld.com/`, b.

J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan. SALSA: Analyzing Logs as State Machines. In *USENIX Workshop on Analysis of System Logs (WASL)*, San Diego, CA, Dec 2008.

J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan. Mochi: Visual Log-Analysis Based Tools for Debugging Hadoop. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, San Diego, CA, May 2009.

Vlasios Tsiatsis, Ram Kumar, and Mani B. Srivastava. Computation hierarchy for in-network processing. *Mob. Netw. Appl.*, 10(4):505–518, 2005. ISSN 1383-469X. doi: http://doi.acm.org/10.1145/1160162.1160174.

Mark Viera. Befriending generation facebook. *The Washington Post*, July 2008.

J. Kulik W. R. Heinzelman and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking*, pages 174–185, Seattle, WA USA, 1999.

Xirrus. 802.11abg wi-fi arrays. http://www.xirrus.us/products/arrays-80211abg.php.

Yahoo! Yahoo Raises Commitment to Cloud Computing with Hadoop. http://bit.ly/ECJHJ.

Tingxin Yan, Deepak Ganesan, and R. Manmatha. Distributed image search in camera sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 155–168, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-990-6. doi: 10.1145/1460412.1460428. URL http://dx.doi.org/10.1145/1460412.1460428.

Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks, 2002. URL citeseer.ist.psu.edu/yao02cougar.html.

Fang Zhiyuan, Chen Xiaoyun, Tang Yong, Zhang Jingchun, and Zhou Yu. Real-time state management in mobile peer-to-peer file-sharing services. *Service-Oriented Computing and Applications, IEEE International Conference on*, 0:255–260, 2007. doi: http://doi.ieeecomputersociety.org/10.1109/SOCA.2007.36.