Mobile Cloud Computing for Data-Intensive Applications

Senior Thesis Final Report

Vincent Teo, vct@andrew.cmu.edu
Advisor: Professor Priya Narasimhan, priya@cs.cmu.edu

Abstract

The computational and storage capabilities of today's mobile devices are rapidly catching up with those of our traditional desktop computers and servers. In fact, mobile phones with 1 GHz processors are readily available in the market today. Unfortunately, all of these processing resources are mostly under-utilised and are generally used to process local data and programs only. With the use of local wireless networks, we can enable these phones to communicate with each other without utilising the resources of a global cellular network. This has the potential to enable collaborative data-intensive computing across a *cloud of mobile devices* without straining the bandwidth of global networks.

To achieve these objectives, Hyrax [3] was initially developed by Marinelli as a MapReduce system [1] that is deployed on a networked collection of Android smartphones. Hyrax is based on Hadoop [4], which is a Java implementation of the MapReduce system and the Google File System [2]. While Marinelli has developed a system that is suitable for initial use to discover the resource constraints/challenges, performance and the scalability aspects of using mobile devices for collaborative data-intensive computing, that initial implementation of Hyrax was not suitable for wide-scale deployment on the mobile devices of common users. To that end, we have improved on Marinelli's implementation of Hyrax, and aim to develop a mobile multimedia share-and-search application that allows users to discover relevant multimedia content on the mobile phones of those within reasonable proximity (ie within the same wifi network). We also evaluate the performance of this new implementation of Hyrax and identify areas for future work, especially in improving the performance and resource consumption (especially power resources) of Hyrax. Furthermore, we also consider the risk that the use of Hyrax poses to the users of the mobile devices that run Hyrax.

1 Introduction

1.1 Motivation

The primary motivation is that of all the processing powers of a mobile device being idle most of the time. However, besides playing the same role as a traditional server, the fact that these devices are mobile give them certain advantages over traditional static servers.

Mobile devices already typically contain a lot of sensors and multimedia data. Hence these data are already on devices within the cloud, therefore uploading them to the cloud would be a simple matter that should not be expensive in terms of resources used. Furthermore, making use of sensor logs and other local data also offer the possibility of performing location-dependent computations and data sharing through colloboration with other devices in their immediate vicinity. Therefore, as opposed to a traditional cloud, the nodes of such a mobile cloud can take advantage of physical location and physicaly proximity to perform computation more strategically. For example, we could envision that making use of sensor logs and other local data offers the possiblity of performing location-dependent computations and data sharing through collaboration with other devices in their immediate vicinity. There mobile devices can communicate with each other directly without making use of a central server, hence avoiding the use of the global network bandwidth (ie Internet / Mobile Data Network). It would therefore be useful to have a mobile cloud computing platform available that allows us to build applications that will enable us to make full use of these mobile resources.

1.2 Hadoop

We only give a brief overview of Hadoop in this section.

Hadoop is made up of two major components: The Hadoop Distributed File System [6] (HDFS, which is similar to the GFS) and the MapReduce system.

A Hadoop job is typically split up into a Map phase and a Reduce phase. Maps are independent tasks that can be run concurrently and independently on different slave nodes, and as the output from these Maps become available, Reduce tasks can begin to process these output to produce the final output for the job. Hadoop is designed to assign tasks to nodes that already contain the data required for the task in order to reduce network transfers.

The HDFS is controlled by the NameNode and various DataNodes, while the MapReduce system is controlled by the JobTracker and various TaskTrackers. Typically the NameNode and JobTracker are run on master nodes (or on a single master node), while each slave node runs an instance of DataNode and TaskTracker each.

The NameNode is responsible for managing the entire HDFS and determining which nodes are to hold blocks for data on the HDFS, while the DataNodes are responsible for the data blocks on their own individual nodes and reporting the status of their individual nodes to the NameNode.

The JobTracker is responsible for distributing tasks (such as Maps and Reduces) to the individual slave nodes, while the TaskTrackers are responsible for actually executing the tasks and reporting the status of the tasks to the JobTracker.

Further information about Hadoop can be obtained from The Apache Hadoop Project website [4].

1.3 Limitations of Initial Implementation

The initial implementation of Hyrax by Marinelli was based on Hadoop 0.19.0 (later patched to Hadoop 0.19.1). However, it made use of features and programs on the Android platform that required root access to the devices, and was therefore not suitable for wide scale deployment on the mobile devices of common users (the assumption being that most users do not have root access to their Android devices).

2 Implementation

A description of how Hadoop 0.21.0 was ported over to the Android OS will be given in this section. We elected to develop the new version of Hyrax for Android 2.1 (or later), for the reason that the API for Android 2.1 is closer to that of Sun's Java implementation. This is important since Hadoop was originally written with Sun's Java implementation in mind, and there are differences in the API available between Sun's Java implementation and that for Android. It would therefore be easier to port Hadoop over to Android 2.1 than to an earlier version of Android since less changes would have to be made to the code. Furthermore, we expect that most users would have at least Android 2.1 running on their mobile phones, considering that the latest version of Android for smartphones is currently Android 2.3. (According to a chart on the Google Developers website, it appears that 93.8% of Android devices that have accessed the Android Market within the last two weeks of the beginning of April 2011 were running versions of Android that were 2.1 or higher [5].) However, many of the function calls made by Hadoop are still not available on the Android 2.1 API (some of them have since been implemented on the Android 2.3 API), and had to be either rewritten or removed altogether.

2.1 Porting Hadoop

Since the primary purpose of the mobile phones was to run as slaves on the Hyrax cluster, we concentrated on porting the relevant code for the DataNode and TaskTracker over to Android. The main challenge of porting the code over remained with fixing the incompatibilities between Sun's Java implementation and the Android API, as well as calls to shell utilities such as `df` which have a different output format from that of a normal Unix distribution.

Three major changes were made to Hadoop during the porting process. The first was to change the configuration files from XML files to properties file, which require less resources to parse [3]. Since the phones wrote configuration files for the jobs in properties files as well, this also required that we also change the code for the master node on the desktop machine to read properties files instead of XML files.

The other major change that was made was in the way Map and Reduce tasks were launched on the slave nodes (ie the phones). Originally, Hadoop launches Map and Reduce tasks by running each task in a new process. As noted in [3], we are unable to execute new JVM instances on the mobile phones due to the lack of the `java` program on the phones. Thus, a call to the main method of the child process is made instead, and the task is executed in a new thread of its own in order to allow the Hyrax app to continue running and accept user interactions.

The final major change that was made was due to the lack of support for executing traditional Java bytecode on Android devices. Hadoop sends tasks to the TaskTrackers through Java classes that are sent to the slave nodes when required. Due to the difference in the format of the bytecode that can be executed on Sun's JRE implementation and Android's runtime system, this system will not work without recompiling the original Java bytecode to the `.dex` format that Android uses, which means that Java classes that were compiled using a Sun compiler will not run on the Android phones. Therefore, all code that Hyrax is expected to execute must be included in the Hyrax package installed on the mobile devices in the current implementation. This ability to dynamically load classes that describe jobs might be supported in a future implementation.

Besides the three major changes above, the concept of a machine name on the cluster was also changed. Since mobile phones do not have machine names as traditional desktop computers and servers do, a change was made to use the phone's IP address as it's name. The individual and unique device IDs of the phones (possibly hashed due to privacy concerns) could also be used as usernames (so the same phone will always be treated as the same user regardless of its IP address, which should not be assumed to be fixed). This will be especially useful when this system is deployed on a proper case study of a multimedia-content search-and-share system.

In addition to the above changes, minor tweaks were also required to the properties of the cluster. As an example, Android only allocates up to 16MB of memory for each application. Therefore, the amount of memory available for sorting map output key/value pairs (`mapreduce.task.io.sort.mb`) has to be greatly reduced from its original default value of 100MB. Also, other properties such as the task timeout and socket write timeout values had to be increase to take into account the slower processing capability of the mobile device as compared to that of a traditional server.

2.2 Cluster Architecture

Our testbed consisted of a cluster of phones (currently 3), a desktop machine acting as the master node (NameNode and JobTracker), and another desktop machine acting as the client. They were all connected to the same router in a private local area network (the phones were connected wirelessly via wifi, while the desktop machines had a physical ethernet connection to the router). Besides installation of the Hyrax application, the phones required no control from the desktop machines.

The HDFS storage space was implemented using the external storage devices of the phones (usually a microSD$^{\mathrm{TM}}$ card). Hence, an external storage device must be present on the phones in order for Hyrax to function. While it is of course possible to augment the cluster with traditional servers that have a lower probability of failing or disconnecting from the cluster, this has not been implemented. Any such implementation will also require the changes described in the previous subsection.

Although the phones were used purely as slave nodes and not clients in the evaluations, our current implementation allows them to act as clients as well, which means that they have the capabilities of starting MapReduce jobs on the cluster. This simply requires that the code for the MapReduce job be included as part of an app, and a mechanism on the user interface of the app to allow the user to start the job. Examples of MapReduce applications that have been ported and tested include the PiEstimator (where the value of π is estimated using a Monte Carlo method), RandomWriter (where random data is generated and stored on the HDFS) and Sort (which sorts data on the HDFS).

3 Evaluation

A huge emphasis will eventually be placed on evaluating the power consumption of the system, since this will naturally be a primary concern of most users. Besides power consumption, we will also perform evaluations on the performance of the Hyrax system, and identify possible components of Hyrax / Hadoop for future optimisation in terms of both performance and resource consumption. The evaluations will be mostly based on the benchmark tests that were used in [3], so that we will be able to compare the performance of this implementation of Hyrax over the previous one. There will be less emphasis in comparing the performance of Hyrax with that of Hadoop on traditional clusters since such a comparison was already performed in [3].

On a less technical side, users will also be naturally concerned about their privacy when using Hyrax, since Hyrax has the ability to read and upload personal data on their individual devices to the HDFS. This issue will be briefly discussed in subsection 3.5.

3.1 Hardware

3.1.1 Slave Nodes

The new implementation of Hyrax was developed using 3 Android G1 (HTC Dream) mobile phones, all running the Android 2.1 “Eclair” operating system. These phones acted as the slave nodes in the Hyrax cluster, and each had a DataNode thread and a TaskTracker thread running on it. The phones were each equipped with a 528 MHz Qualcomm MSM7201A processor, 192 MB of RAM, a 1150 mAh lithium-ion battery and IEEE 802.11b/g connectivity [7]. They were each also equipped with a microSD$^{\text{TM}}$ memory card (one with a capacity of 1 GB, the rest with a capacity of 8 GB). Although the development was initially tested on phones that had root access, the development was done with devices that do not have root access in mind. Tests will also be performed on phones that do not have root access when those became available at our disposal to ensure that our implementation did not make use of any features or programs that required root access.

3.1.2 Master Node

The master node was run on a traditional desktop computer running Ubuntu 10.10. This node had the NameNode and the JobTracker running on it. The machine is equipped with an Intel® Pentium® D Processor 830 processor and 2 GB of RAM. Since the desktop machine was only used to run the NameNode and JobTracker, the computational and memory load placed on it was not too significant, and it is expected that the specification of its hardware will not have much impact on the performance of the Hyrax cluster.

In addition to the master node that ran the NameNode and JobTracker, another desktop machine was used in the evaluation process by acting as the client, starting MapReduce jobs that were executed on the phones in the cluster.

3.2 Execution Time

We ran a few benchmark jobs aimed at collecting some statistics about the performance of Hyrax in terms of its execution time. To get a sense on which task type (map, reduce, shuffle / sort) takes the longest in terms of execution time (and is hence a possible limiting factor in the performance of Hyrax jobs), we first ran a simple WordCount job (where the number of occurrences of each word in a text document is counted) on a single node. The input file was a text document that was approximately 3.14 MB in size. The results are shown in Figure 3.2.

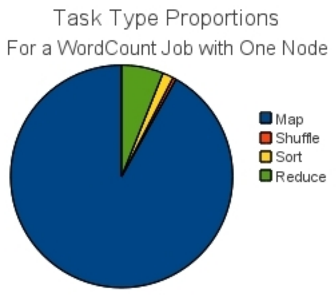

Figure 1: Graph of Proportion of Execution Time spent in each Phase of a WordCount Job on One Node

The graph suggests that the map phase of a MapReduce job appears to be phase that takes the longest time to execute. Of course, this also depends on the type of job that is being executed. As this was a one-node job, not much time was spent in the shuffle phase. We expect that this would increase if more nodes were added to the cluster. A small proportion of time was also spent in the sort phase. This was expected since the output, which was quite substantial, had to be sorted prior to being sent to the reduce function. Overall, however, it appears that the map phase of the job overwhelmingly took up the most of the execution time for the job.

We also tried to establish a link between the total execution time of a job and the number of nodes in the cluster. In order to do so, we ran a RandomWriter job, in which 2 map tasks are executed on each node, each writing 1 MB of random data to the HDFS. This job is similar to the one in [3]. The results are shown in Figure 3.2.

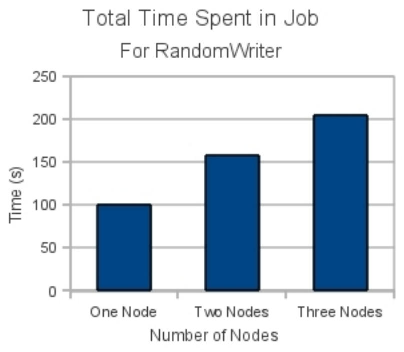

Figure 2: Graph of Total Execution Time against Size of Cluster for RandomWriter

The results suggest that there seems to be some overhead involved when more nodes are added to the cluster, since the total execution time appears to increase by about 50 seconds for each new node added. We would expect that the increase in task execution time would be less pronounced as even more nodes are added to the cluster. In comparison to [3], it would appear that this newer version of Hyrax has significantly more overhead involved as the total execution time for a similar job is higher in this case. Furthermore, we also collected data on the amount of time that was spent actually executing the map phase (there is no reduce phase in this job). The data is shown in Figure 3.2.

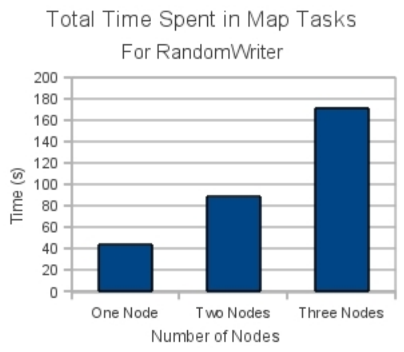

Figure 3: Graph of Time Spent in Map Phase against Size of Cluster for RandomWriter

Note that the total time spent in the map phase does not correspond to the total execution time for the entire job since some of the map tasks were actually executed in parallel. However, the data in this graph shows that the total time spent in the map phase actually corresponds closely with the total execution time presented in [3]. This further suggests that the amount of time spent in overhead (initialisation, data transfer etc) in the new version of Hadoop (0.21.0) and Hyrax appears to be significantly more than that of the older version (0.19.1).

3.3 Resource Usage

We monitored the CPU utilisation during the execution of the WordCount job and the result is shown in Figure 3.3.

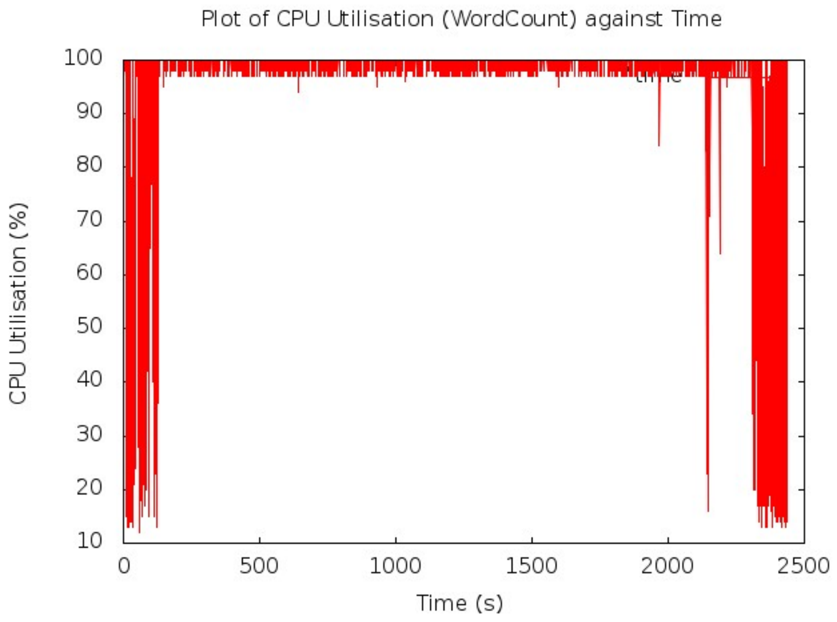

Figure 4: Graph of CPU Utilisation against Time for WordCount

This graph suggests that the CPU utilisation is very high when the WordCount job was executing (the beginning of the graph was the CPU utilisation before the job was started, and the end of the graph was the CPU utilisation after the job was finished). This has serious consequences for the usability of the device when a Hyrax job or task is being executed on it. This data, however, corresponds to that from the earlier study in [3].

3.4 Power Consumption

The power consumption of Hyrax was evaluated by running the same RandomWriter and Sort jobs in [3] repeatedly until the battries in the devices were depleted. Readings of the battery level of each devices were taken periodically. We have performed the experiment on three mobile phones, and plan to extend that to five and / or seven when possible so that we can determine the impact on power consumption that the number of nodes involved in a job will have. We also plan to determine how much of the battery power were used in the Map / Reduce phases of a job. The results of this analysis could provide a future direction when working on optimisations for Hyrax.

The result of running the experiment on three phones is shown in Figure 3.4. As can be seen, the battery is capable of running for over 5 hours, which is comparable to the previous version of Hyrax, where the batteries lasted for about 6 hours.

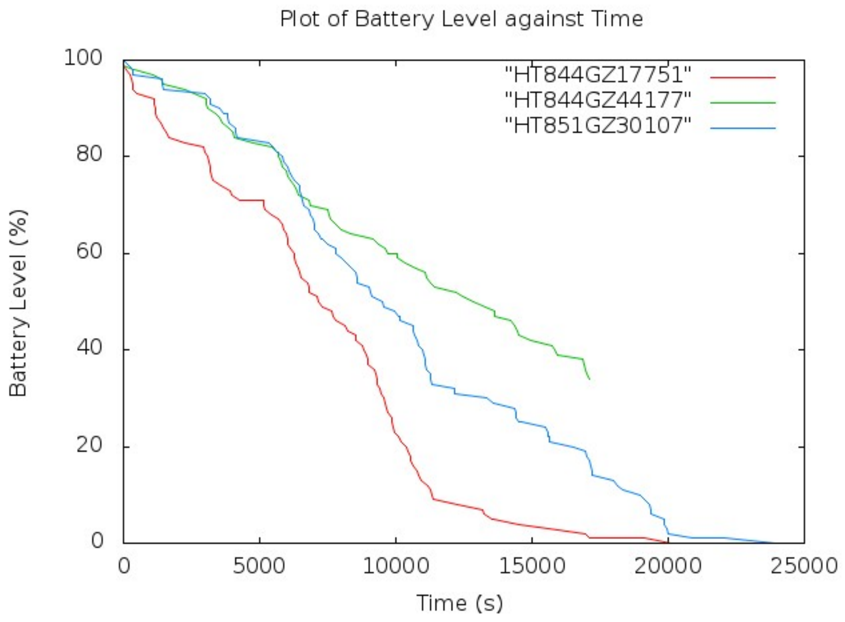

Figure 5: Graph of Battery Level against Time for a 3 Node Cluster running RandomWriter / Sort

3.5 Privacy Concerns

Privacy concerns are important to all users, and Hyrax aims to achieve its goals of enabling mobile cloud computing without compromising on the privacy of its users. Since Hyrax is designed to run on the mobile devices of its users, the most obvious concern is whether any of the user's personal data on the device will be exposed to the rest of the cluster. As dynamic class loading is not currently implemented in Hyrax, it is not possible to execute code on the phones that are not already part of the code of the Hyrax app. Therefore, as long as the Hyrax application itself does not contain any code that will upload local data to the HDFS without the user's expressed permission, the local data of the device will not be visible on the HDFS. Of course, this is contingent on the security of Hadoop itself, but we feel that the risk of this is no more than any other application that is available on the Android market. However, this will have to be reconsidered this when dynamic class loading is enabled in the future. Possible mitigations include restricting what the code that users can call in the `map` and `reduce` functions.

Another possible concern stems from the use of sensor data, which could possibly give away the location of a device. While this is not a concern in the current implementation, we envision that such sensor data will be used extensively in the future in a variety of applications (possible examples include monitoring traffic or tagging pictures etc). One possible solution is to annonimise the name of the source of the data. Another is to not record the source of the data at all. However, such privacy concerns can and should be further explored and resolved depending on the application.

4 Test Study

We aim to develop a mobile multimedia share-and-search application that will allow user to upload photos from their mobile devices onto the cluster (HDFS). This will allow the other uses in the cluster to search for and download these photos onto their own devices. For example, users could possibly run a Hyrax job that searches through every single photo on the HDFS to find those that contain a certain face that the users provide to the program. This application is currently still in development. It currently has the capability of detecting faces in a photograph and storing those faces along with the original photograph in the HDFS. However, one major obstacle to processing photographs on a mobile device is the amount of memory usage involved.

In order to ensure that we do not exhaust the memory allocated to the process, it is necessary for our application to first reduce the size of the images before performing any processing on them.

5 Conclusion

We have presented our implementation of a mobile cloud computing platform on mobile devices running the Android OS. While our implementation works, it appears that much still needs to be done to improve the performance of Hyrax. Possible critical areas of improvements would include the overhead involved in executing a Hyrax job, and also the resource utilisation during the execution of a task on a device (especially the CPU utilisation). This is critical in ensuring that the user of the usability of the device is not sacrificed when jobs are being executed.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *USENIX Symposium on Operating Systems Design and Implementation*, San Francisco, CA, Dec 2004, pp. 137 - 150.

[2] S. Ghemawat, H. Gobioff and S. Leung, "The Google file system," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, Oct 2003, pp. 96 - 108.

[3] E. Marinelli, "Hyrax: Cloud Computing on Mobile Devices Using MapReduce," *Master's Thesis, Technical Report CMU-CS-09-164*, School of Computer Science, Carnegie Mellon University, 2009.

[4] The Apache Hadoop Project. `http://hadoop.apache.org`

[5] Google Developers. Platform Versions. `http://developer.android.com/resources/dashboard/platform-verions.html`

[6] The Hadoop Distributed File System Architecture. `http://hadoop.apache.org/hdfs/docs/current/hdfs_design.html`

[7] HTC. HTC Dream specification. `http://www.htc.com/www/product/dream/specification.html`