DESIGN AND IMPLEMENTATION OF DISTRIBUTED MOBILE COMPUTING PLATFORM USING HADOOP

BY SHRADDHA PANDHE

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Professor Dipankar Raychaudhuri

and approved by

_____

_____

_____

New Brunswick, New Jersey

October, 2013

ABSTRACT OF THE THESIS

Design and Implementation of Distributed Mobile Computing platform using Hadoop

by Shraddha Pandhe

Thesis Director: Professor Dipankar Raychaudhuri

This thesis is aimed at design and evaluation of a distributed cloud computing platform using mobile nodes connected via cellular network. Such a Mobile Cloud capability is motivated by situations in which the cellular access network has limited backhaul bandwidth to the Internet as may be the case in developing regions or disaster recovery scenarios. In view of the growing importance of the OpenStack and Hadoop/MapReduce framework, we first evaluate the performance of Hadoop in the mobile cloud environment and then consider modifications necessary to improve performance. The study was conducted using an experimental methodology based on the GENI open WiMAX base station at WINLAB, which was used to set up a cluster of 4G/cellular mobile clients running Hadoop. The results show that as expected, performance of Hadoop cloud applications is severely degraded due to mobile device channel quality variations in the baseline case, but that significant improvements (5x in run-time) can be realized with relatively modest cross-layer (i.e. base-station and signal-to-noise ratio) aware strategies. The results also show that, there are certain types of applications that are not suitable for Distributed Mobile Computing, e.g. applications that generate large outputs, while applications with small sized output benefit greatly from Mobile Cloud.

Acknowledgements

I would like to thank, firstly, my family for their unwavering support and constant encouragement through my graduate study. I would like to thank my advisor Prof. Dipankar Raychaudhuri, for his invaluable guidance and advice while working on my research. His inspiration was a driving force in many cases. The numerous discussions held with him have given me a deeper understanding of a lot of topics involving both wireless networks, system level architecture and countless other areas as well. I would like to express my gratitude to Prof. Wade Trappe and Prof. Rich Martin for being on my thesis committee. I am grateful to all my friends at Winlab and the ECE Dept. as well, without whose invaluable help this experience would not have been the same. In particular, I would like to thank Ivan Seskar and Kishore Ramchandran for their encouragement and support throughout this research project.

Dedication

To my parents, brother, grandparents, my teachers and friends

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

A distributed system consists of multiple computing machines (a.k.a nodes) that communicate via a network. These nodes interact with each other in order to achieve a common task [1]. In commercial deployment, Google uses thousands of commodity servers interconnected with high bandwidth links, to serve requested web pages to millions of users. These pages are stored in Google File System (GFS) [2], a distributed file system, that supports data replication, similar to Hadoop Distributed File System (HDFS) [3].

Distributed Mobile Computing (DMC) builds on top of this architecture, wherein, a collection of mobile devices (a.k.a cluster) collaborate to accomplish a complex task. There are several reasons why it is judicious to use mobile devices for distributed computing. Firstly, of late, the abundance of smartphones has started growing exponentially. International Data Corporation (IDC) reports that, 712.6 million smartphones were shipped globally in 2012, which was 44.1% more than in 2011 [4], while there was a small decrease in the worldwide shipment of server units by 1.5%, to 8.1 million units in the year 2012 [5]. Hence, rather than relying on the small-volume server market, we can leverage the massive volume economics of the smart device world driven by cell phones and tablets. Secondly, current mobile devices exhibit excellent price/performance as compared to the current generation server CPUs. Less competition and high demand in server processor market has lead to far higher prices ($100 to $1000) and less price/performance, as compared to the mobile devices ($0 to $200). Although this would imply less computing power per node, there are several types of workloads that can still benefit greatly from DMC. Some workloads do not require best

performing processors to achieve their Service Level Agreements (SLA), while some other workloads are networking, storage or memory bound. Hence, using high power server CPUs will accomplish very little for these applications. Lastly, mobile devices have evolved tremendously in terms of power efficiency, as battery life has always been a crucial parameter for portable devices. Typical servers consume 200W to 500W of power while mobile devices use 10W to 20W. Hence power/performance of portable device is much better than servers.

But, there are several hurdles that complicate the process of building such a system. Mobile devices communicate over the wireless medium, which is inherently slower than wired and even the evolving cellular technologies like 4G cannot match up to the speeds of Gigabit Ethernet. Also, the nodes' availability in the cluster is not guaranteed as they might enter a no-network zone (e.g. tunnels, elevators etc) at any time. To add to it, the data throughput, and depends on factors such as active users and signal-to-noise ratio (SNR). The SNR of a link depends on the distance of the node from the transmitter base station, hence random mobility pattern may lead to variable delays, resulting in unpredictable performance. Hence, mobile network is not readily suitable for distributed computing and needs a robust overlay platform.

Of several distributed computing platforms, Hadoop [6] suits most to our requirements. With Hadoop Distributed File System (HDFS) [3] and MapReduce [7] as its building blocks, Hadoop, to some extent, provides the necessary robustness needed for DMC. HDFS is a distributed, scalable, and portable file system. To guard against machine failures, it replicates every file on multiple machines. The default replication factor of three implies that at any given time, each file will be available on three different nodes in the cluster. HDFS also re-replicates a file when the total number of replicas fall below the associated replication factor (due to failure). This provides the necessary robustness required in the distributed mobile computing architecture. Later chapters of this thesis describe in detail the how, with certain modifications, Hadoop can help to solve the challenges associated with DMC.

1.2 Problem Statement

The aim of this thesis is to study the nature of a cellular network to analyze the hurdles it puts forth for DMC, and to build a robust overlay platform based on an improved hadoop implementation

1.3 Methodology

The study of the dynamics of cellular network, has been done using Worldwide Interoperability for Microwave Access (WiMAX) [8] system provided by Wireless Information Networking Laboratory (WINLAB) [9] research facility. We also studied the Hadoop ecosystem and its components. The inspection of the design strategies of Hadoop lead to a conclusion that, as is, Hadoop is inefficient for a wireless environment. We identified the enhancements that need to be made in order to support cellular network. All experiments were performed with WiMAX and ubuntu-12.04 servers available at WINLAB.

1.4 Contributions

Specifically the contributions of this research project are as follows:

1.4.1 Design and implementation of Base Station aware HDFS (BAHDFS)

HDFS relies on the principle of rack-awareness for replica placement. As explained later in the chapter 3, this strategy has some pitfalls with respect to the wireless network. In the proposed approach we define Base Stations as Racks as and Mobile Devices as Servers. The idea is to prefer nodes on different base stations for replica storage, so as to avoid sharing of single downlink channel.

1.4.2 Design and implementation of SNR and Load Based Node Selection system (SLBNS)

In a practical scenario, the cluster of mobile devices consists of several scattered nodes across the coverage area of the base station. The SNR of a wireless link is inversely proportional to the distance from the base station. Therefore, selecting nodes with better SNR can have great impact on the performance of MapReduce job. In this contribution, we have designed and implemented a mechanism to select nodes in the order of their SNR, while assuring not to overload the nodes with high CPU usage.

1.4.3 Design and implementation of Unpipelined HDFS

The aim of this thesis is to study the nature of a cellular network to analyze the hurdles it puts forth for DMC, and then get rid of the inefficiencies of Hadoop in lieu of wireless environment to build a robust overlay platform. In current implementation, when a client wishes to create a file in HDFS, for each block of that file (depending on file size) it requests the NameNode for a list of potential datanodes (= replication factor). The NameNode returns a list of datanodes, in the increasing order of their distances from the client. The client then forms a pipeline, where, it sends the data-block to one of the nodes, and requests it to forward it to other nodes in the pipeline. As we will see the chapter 3, even though this strategy suits for wired environment, it performs very inefficiently in wireless environment. In this contribution, we have removed the pipeline and modified the client to send the data to all the nodes in the list by itself, instead of pipelining it

Chapter 2

Background & Related work

2.1 Background

2.1.1 Wireless Interoperability for Microwave Access

WiMAX is an IP based technology which provides wireless broadband access with performance similar to 802.11/Wi-Fi networks, but with coverage and Quality Of Service (QoS) of a cellular network. WiMAX is intended to be a wireless digital communication system, also known as IEEE 802.16, a standard intended for Metropolitan Area Networks. WiMAX is designed to support both stationary as well as mobile stations for broadband wireless access. For stationary, WiMAX can provide services upto 30 miles radius, whereas for mobile stations, this range reduces to 3-10 miles. To compare with 802.11/Wi-Fi, Wi-Fi standard is limited in most cases to only 100 - 300 feet, whereas WiMAX is intended to support comparable data rates, but with cellular coverage [8]. The amount of throughput that can be delivered to a customer radio from a base station is always inversely proportional to range, i.e. higher the range, lower the bandwidth.

A WiMAX radio contains both, a transmitter and a receiver. The carrier frequency used in WiMAX is between 2 GHz to 11 GHz. The first International standard that came in was of 3.5 GHz spectrum. United States has provided 5.8 GHz free spectrum which has attracted a few vendors over the years. Licensed spectrum at 2.5 GHz used both domestically in the US and fairly widely abroad is the largest block in the US. Also, in the US and in Korea products are shipping for the 2.3 GHz spectrum range. citewimax

As a part of the research project, an extensive study of the performance of wireless network was conducted. It was necessary to analyze and quantify the performance of

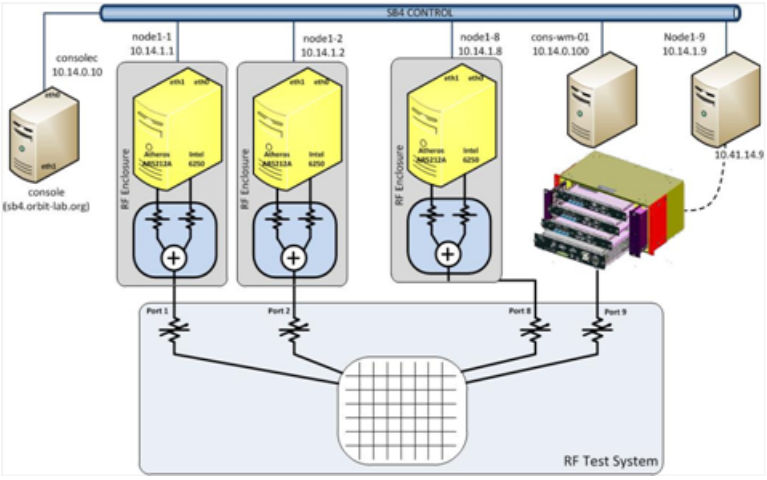


Figure 2.1: Transceiver Test System

WiMAX.

A wireless link is characterized by properties like Throughput and Latency.

Throughput: Every base station specifies a theoretical peak throughput. The base station has different uplink and downlink data rates that depend on the alloted uplink/downlink ratio. The theoretical data rates are not observed in practice due to several reasons like interference, sharing, SNR etc. Hence, the actual throughput can be much lower than expected. The experiments were designed to study this effect in detail.

Latency: Latency is another important factor contributing towards data rate. Latency can refer to several kinds of delays incurred during processing of network data. A low-latency network typically shows low delays, whereas a high-latency network shows high delays. Although the theoretical peak throughput of a channel is fixed according to the technology used, the actual throughput varies over time and is affected by high latencies. Excessive latency creates bottlenecks that prevent data from filling the network pipe, thus decreasing effective bandwidth. The study of WiMAX is mainly focused on analyzing above mentioned attributes for various scenarios and topologies.

WiMAX Setup at WINLAB

A sandbox at WINLAB with WiMAX capabilities was used for the study as shown in Fig. 2.1. More information about the setup can be found at [10] This sandbox consists

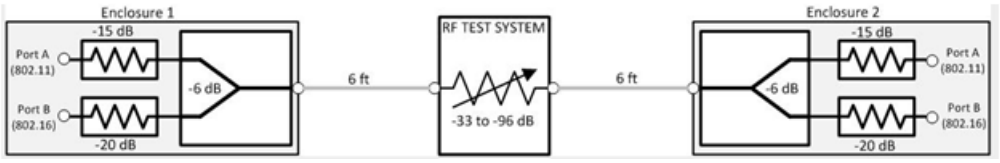


Figure 2.2: Node to node RF path

of 9 nodes and a WiMAX subsystem. The subsystem includes a base station and a controller. Each node is enclosed in an RF enclosure which provides 80dB of isolation. There are total 8 nodes as wireless devices (with Atheros 5000X mini-PCI card and Intel 6250 mini-PCIe 802.11/802.16). These nodes are connected to a power combiner and fed through the enclosure access port to the outside connector which is wired to one of the input/output ports of the RF Transceiver Test System. This configuration enables fairly tight control of RF attenuation matrix between the nodes. There are total 9 input/output ports on the transceiver test system as seen from Fig. 2.1. The initial 8 ports are connected to 8 nodes of the sandbox and 9th port is connected to the base station.

Node to node RF path:

As can be seen from the Fig. 2.2, the two nodes are housed inside the enclosure to avoid interference. There is an attenuator system in between two nodes, which allows us to calibrate the power level even further. The matrix essentially helps in simulating a more real life scattered topology with all nodes at different and random distances from the transmitter. The level of attenuation corresponds to the distance from the base station, and link quality in turn depends on this distance. As these type topologies are very common in real life, studying their behavior was a crucial part of this study.

The goal was to completely characterize WiMAX behavior and extensive set of experiments were performed on the test bed. The experiments started with evaluation of individual WiMAX links and slowly moved to more complicated scenarios to simulate real life topologies. For all these experiments, sandbox 4 [**?**] was used. Before starting the experiments, the setup was tuned to optimum performance. Following parameters needed to be changed in order to adjust the system.

- Maximum Received Power at the base station: Modified from -45dB to -25 dB

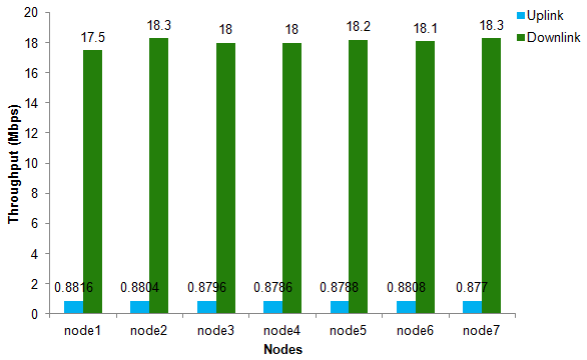


Figure 2.3: Throughput evaluation of all nodes using TCP

- TCP Receive buffer: Modified from 87.5 KB to 3.77 MB to gain highest throughput from a link, it is necessary to fill the network pipe. One way to achieve this is to set receive buffer size equal to delay-bandwidth product [11]

Evaluation of individual links The following assumptions were made for the evaluation of individual links of all nodes

- Aim: Evaluate uplink and downlink throughputs for all the nodes in the cluster using TCP

- Setup: Nodes - 7; Modulation Scheme - Downlink: 64QAM 3/4, Uplink: QPSK 1/2; Tool: iperf

- Results: The scenario where single node actively downloads data from the Base Station was the best candidate to analyze the performance of the link. Fig. 2.3 shows the individual link evaluation of all the nodes in the test setup. Similar experiments were conducted for UDP and those results were used as the baseline. UDP serves as ideal baseline because it provides the value of highest effective throughput, as it omits the negotiation phase. The target was to achieve comparable throughput for TCP. After tuning a few parameters such a modulation scheme and TCP Buffer size, the throughput was optimized to be 18Mbps which was equal to the observed values for UDP.

As evident from Fig. 2.3, uplink throughput values are significantly lower than downlink, which can be attributed to following reasons.

- The modulation scheme used for Uplink (MCS: QPSK 1/2), which has 4 bits/symbol, whereas Downlink (64QAM 3/4) uses 6 bits/symbol.

- A single WiMAX frame consists of 47 slots (symbols). The channel is time-shared among these symbols. The downlink/uplink ratio used in the experiments is 35:12. i.e., in a single frame, out of 47 slots, 35 slots are allocated for downlink, and remaining 12 slots are for uplink. Thus the downlink rate is at least 3 times that of uplink.

Analysis of the effect of channel sharing

- Aim: In distributed mobile computing, number of mobile devices communicate and synchronize with each other to complete distributed task. To acheieve this, all these devices need to exchange data on a shared channel. This experiment was designed to analyse the behavior in this scenario.

- Setup: Nodes: 7; Modulation Scheme - Downlink: 64QAM 3/4, Uplink: QPSK 1/2; Tool: iperf

- Results: From Fig. 2.4 it can be seen that, when nodes transmit and receive simultaneously, effective throughputs are much lower than individual due to the time shared channel. Theoretically, total throughput should be equally divided among all the nodes, because all the slots in one frame will be shared uniformly among them.

The theoretical effective throughput is,

$$Effective\ Throughput = \frac{Total\ Throughput}{Number\ of\ active\ nodes} = \frac{18Mbps}{7} = 2.57Mbps \quad (2.1)$$

As seen, the observed throughput is between 2.5 Mbps to 3 Mbps, which is close to theoretical value.

As evident from Fig. 2.5, uplink throughput in case of channel sharing is extremely poor. We will see in later chapters that, this is a very significant issue limiting the type of applications that we can run on mobile devices. This result implies that, the applications will timeout if all the nodes start uploading the data at same time. Situation

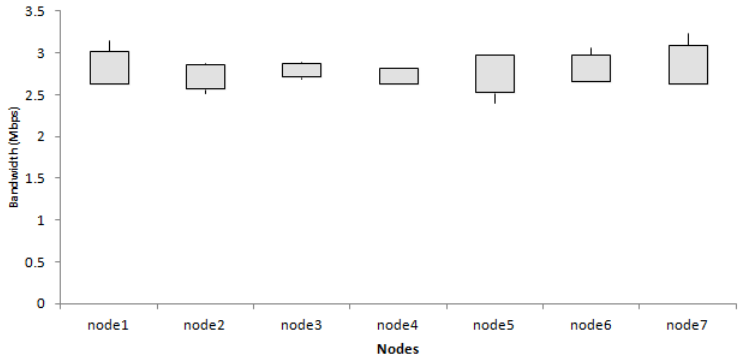


Figure 2.4: Multinode Downlink Throughput for TCP

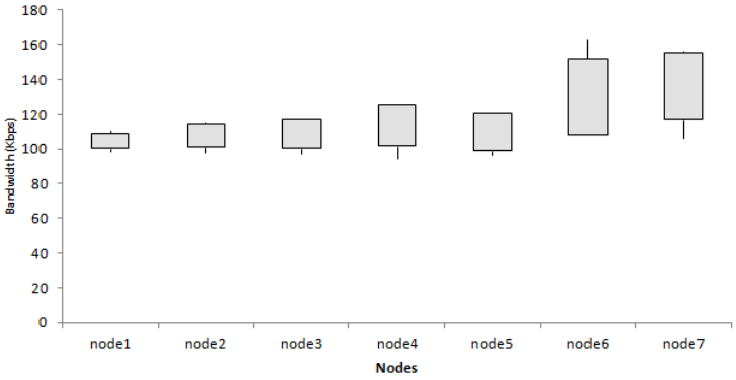


Figure 2.5: Multinode Uplink Throughput for TCP

would worsen if the data size is large. Hence it would better to avoid making a cluster out of all the nodes under same base station.

Study of Latency

- Aim: Latency is an important factor to consider with respect to distributed mobile computing.
- Setup: Nodes: 7; Tool: Hping3. The latency is calculated by transmitting a small packet across the network, and calculating the round-trip-time (rtt). We used the hping3 tool to send relatively small sized TCP packets.
- Results: Table 2.1 shows the observed TCP latencies in this setup.

Study of various Modulation and Coding Schemes (MCS)

Table 2.1: Latency evaluation for TCP

Node	Average latency (ms)
Node 1	80.53
Node 2	80.488
Node 3	80.458
Node 4	81.278
Node 5	80.296
Node 6	79.476
Node 7	83.256

- Aim: Study of various modulation and coding schemes
- Setup: Nodes: 1; MCS: 64QAM 3/4 through QPSK 1/2
- Results: Fig. 2.6 shows the throughput evaluation for uplink and downlink at different MCS

WiMAX supports a variety of modulation and coding schemes and allows for the scheme to change on a burst-by-burst basis per link, depending on channel conditions. The higher order modulation schemes result in higher data rate. More complex modulations require better link conditions such as less interference and good line of sight. Coding scheme is an indication of how much of the data stream is being used to transmit usable data. This is expressed as a fraction with the most efficient rate being 5/6 or 83.3Modulation and Coding Scheme (MCS) is the term used to describe the combination of Modulation Scheme, such as QPSK through 64QAM; and the Coding Scheme, such as rate 1/2 through 5/6, used when transmitting data. As depicted in Fig. 2.6, each MCS has an associated data rate, and higher order modulation schemes (e.g. 64 QAM) show much higher data rates than the lower order ones (e.g. QPSK).

Study of effect of attenuation on throughput

The Signal to Noise ratio (SNR) of the link between a mobile device and the base station depends upon the distance between them. Larger the distance, lower the received signal strength (due to higher attenuation), and lower is the SNR. When the SNR is small, the base station tries to reduce the data rate and increase the correction bits to reduce bit error probability. i.e. the base station tries to shift from higher order

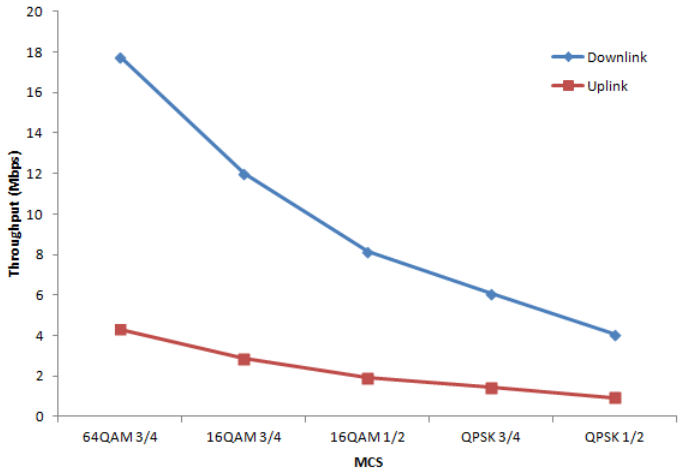


Figure 2.6: Study of various Modulation and Coding Schemes

modulation scheme to the lower ones in order to reduce error rates and avoid retransmissions. Hence, farther devices end up getting lower bit rates. In the distributed computing scenario, it is crucial to understand this behavior, as the mobile nodes move frequently causing unpredictable link qualities. This link quality analysis drives the data placement strategy for HDFS in wireless environment. i.e. the devices with better link quality are preferred over those with worse link qualities. Even though the topology changes over time, it is presumed that, a mobile device would not participate at all if it is not idle or lightly used at the time. Hence it can assumed the device is going to be stationary during the execution of at least one task, which is less than 30 minutes. The test setup that is used in our experiments is stationary, i.e. the distance between node and a base station is always constant. Hence, RF attenuation matrix is used to simulate mobility.

- Aim: To analyze the behavior of throughput when the node is in motion i.e. when the attenuation is varied.
- Setup 1: Single node, Attenuation: 0dB to 30dB
- Results 1: As seen in Fig. 2.7, as the attenuation increases, the link quality degrades, causing effective throughput to drop. This plot shows how a link behaves as it moves away from the base station. As the attenuation increases, the base station appropriately switches to a different (lower) MCS, in order to cope up with

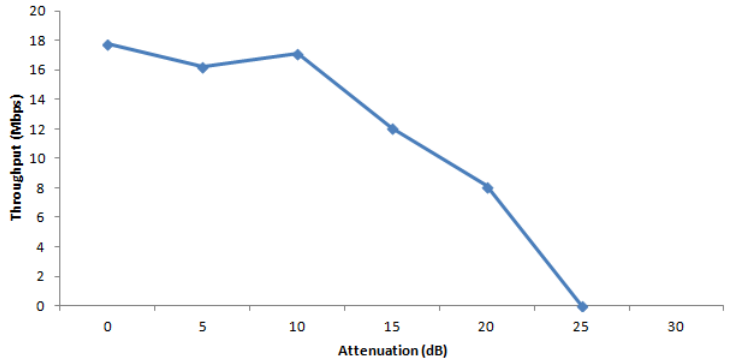


Figure 2.7: Effect of attenuation on throughput (Simulation of a moving node)

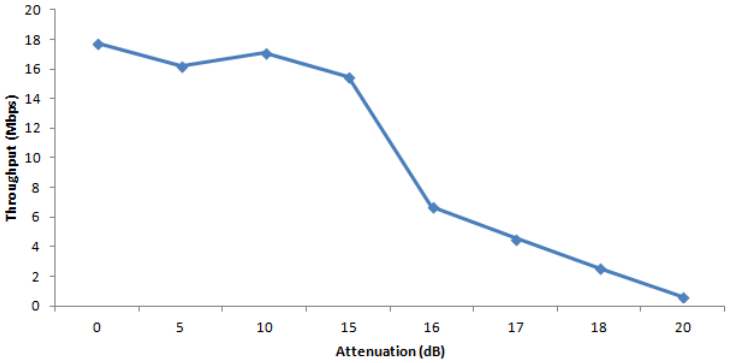


Figure 2.8: Effect of attenuation on throughput (MCS: 64QAM 3/4)

the degrading link quality, as higher MCS cannot sustain at farther distances. This trend is described comprehensively in Fig. 2.8 and Fig. 2.9

As shown in Fig. 2.8, when MCS is kept constant at 64QAM 3/4 and attenuation is varied from 0dB to 30dB, the throughput drops to 0 at mere 20dB. This is because, at higher attenuations, bit-error probability increases significantly. This scheme is not designed to be very robust to deal with poor link as it does not allocate sufficient number of bits for error correction to sustain at such higher error rates. Hence, the performance plummets to 0 very sharply. Hence, to sustain in such conditions, the base station switches to a more robust, but slower modulation scheme. This switching of MCS happens throughout the attenuation spectrum. It is evident from Fig. 2.9 that, when all MCS are plotted against attenuation, the default behavior tends to follow an

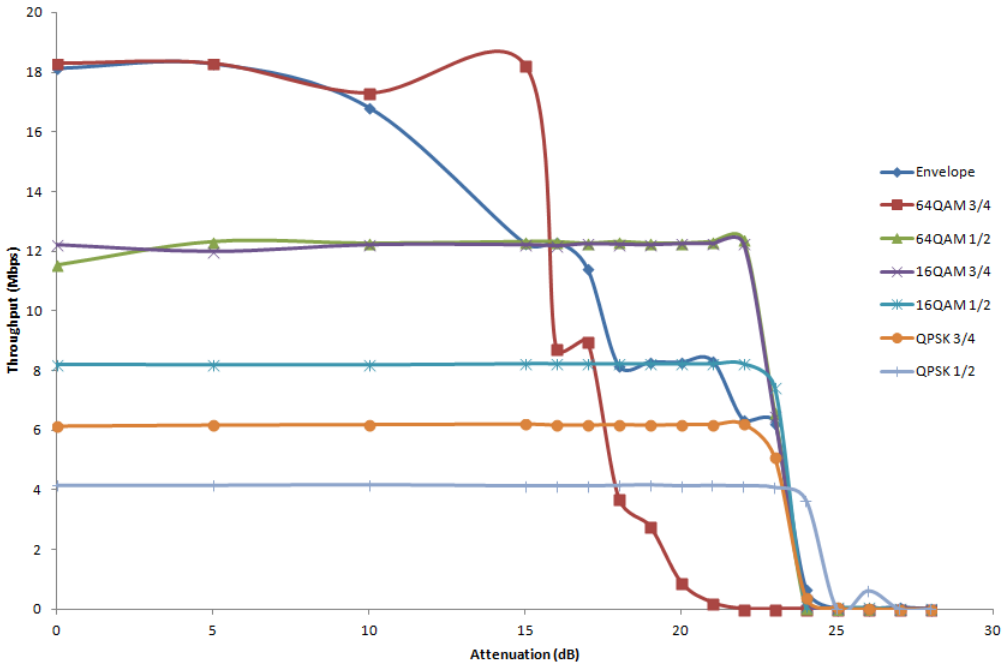


Figure 2.9: Effect of attenuation on throughput (Behavior of different MCS at varying attenuation)

envelope of all those curves.

Study of Proportional Fairness scheduling algorithm

The base station uses scheduling algorithms to effectively share the channel among all the users. The objective of every scheduling algorithm is to optimize one or the other parameter of the channel. There are some schemes that are simple and fair, but not aware of the Quality of Service (QoS), while there are other schemes that are complicated and unfair, but provide QoS guarantees [12].

Round-Robin (RR): This is a very simple algorithm specifically designed for time sharing system. In this scheme, irrespective of the priority, the scheduler assigns time slots to each queue in equal proportions. Hence, in one round, once a queue is served, it is not served again until all other MSs in the system have been served. Because of equal time slots, this scheme tends to provide fair scheduling, at the same time compromising on channel capacity by not utilizing to the fullest. Also, it can be easily observed that this scheme cannot provide any QoS guarantees.

Proportional Fairness (PF): This relatively complicated algorithm tries to attain a

balance between two competing interests [13]:

- Optimizing overall throughput
- Providing a minimal level of service or fairness to all the users.

As against RR, this scheme maximizes frame utilization, resulting in highest utilization of channel capacity. As this scheme is used in all the experiments, it was investigated in detail to understand its behavior.

- Aim: To inspect the behavior of Proportional Fairness Scheduling Algorithm.
- Setup: 6 Stationary nodes 1 moving node (Attn: 0dB to 30dB); MCS: Default (64QAM 3/4 to QPSK 1/2)
- Results: As explained previously, in this experiment, only one node out of seven is moving (i.e. varying attenuation), and rest of the nodes are stationary and very close to the BS (i.e. constant 0 attenuation). From FIg. 2.10 it can be inferred that, as one node moves away from the BS (i.e. its link deteriorates), the BS tries to provide fair throughput by maximizing the average rate. The BS tries to boost the rate for the weaker node, at the same time compromising rates for all other nodes. This concept can be explained mathematically as follows. Total maximum downlink throughput (0 dB attenuation):

$$maxThroughput = 18Mbps \tag{2.2}$$

Hence, when all (7 in this case) nodes start actively downloading data, total throughput will be shared uniformly among them. Hence,

$$netThroughput = \frac{maxThroughput}{numNodes} \tag{2.3}$$

$$netThroughput = \frac{18Mbps}{7} = 2.57Mbps \tag{2.4}$$

This netThroughput corresponds to the rate in Fig. 2.11 at 0dB attenuation. Let's consider the point in Fig. 2.10 where attenuation for Node 1 is 20dB. Referring

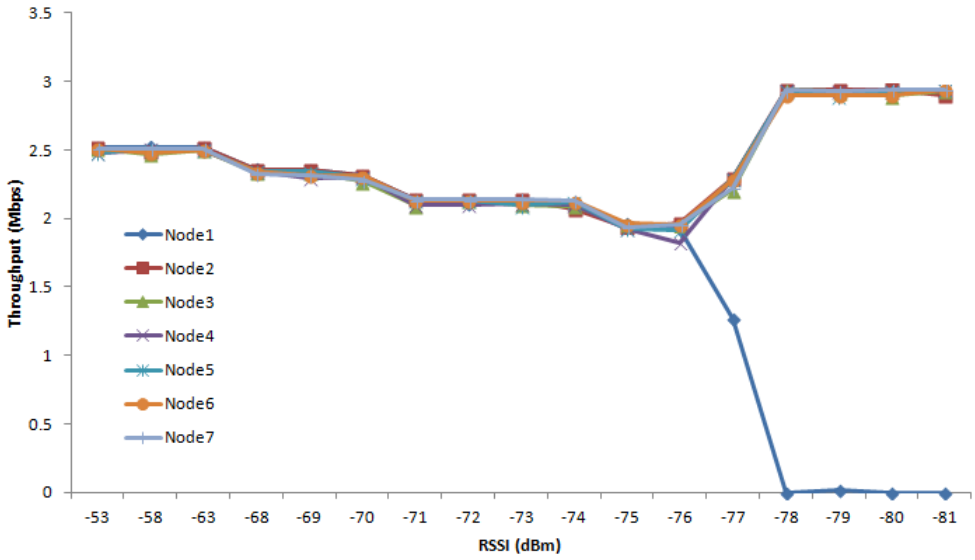


Figure 2.10: Effect of Proportional Fairness algorithm on throughput-1

to the Envelop in Fig. 2.9, this attenuation translates to effective rate of 8Mbps (assuming this is the only active node in the cluster), and if all the nodes are active at same attenuation, effective rate would be 1.14 Mbps. As all the other nodes are at 0dB, from Fig. 2.10, they would receive a rate of 18Mbps (assuming only one of them active one instance), i.e. 2.57 Mbps per node. Hence, average throughput:

$$avgThroughput = \frac{1.14 + 2.57 * 6}{7} = 2.365Mbps \tag{2.5}$$

Corresponding data point in Fig. 2.10 confirms this value.

It is very interesting to see what happens when Node 1 moves so far, that it disconnects from that BS. As shown in Fig. 2.10, once the weak node disconnects, average throughput per node suddenly heightens up. This can be explained mathematically as follows. Once the weakest node disconnects, the numNodes (number of active nodes) becomes 6. Hence from eq. 2.3

$$netThroughput = \frac{18Mbps}{6} = 3Mbps \tag{2.6}$$

This suggests that, its better to skip weaker nodes as far as possible, as they tend

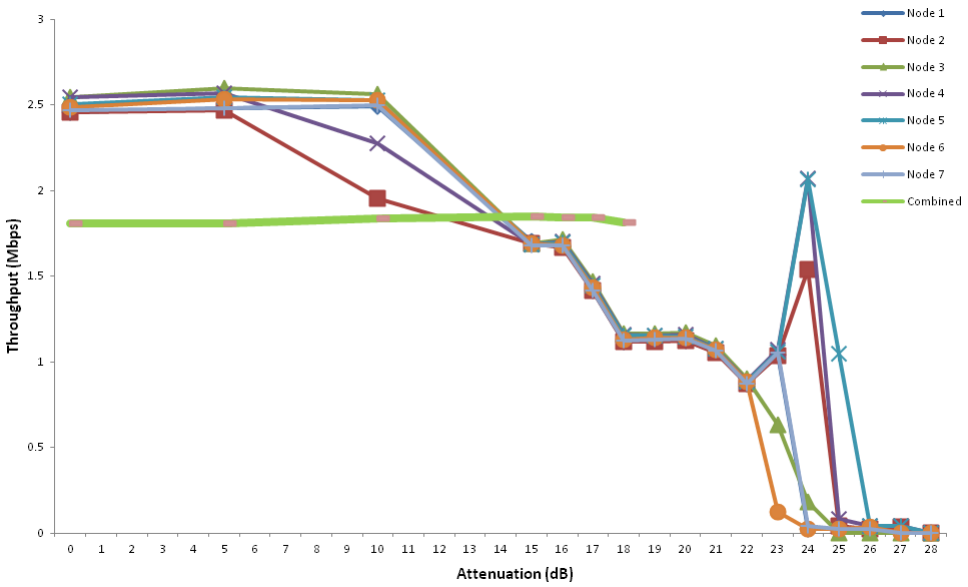


Figure 2.11: Effect of Proportional Fairness algorithm on throughput-2

to bring down the overall throughput. Fig. 2.11 depicts (series Node 1 to Node 7) the same scenario more generally, i.e. the effect on throughput when all the nodes gradually start moving away from the BS. As all the nodes are getting weaker in link quality, the drop in throughput is steeper than Fig. 2.10, but after 22 dB, some of the nodes start disconnecting, as a consequence, enhancing data rates for remaining nodes. In practical environment, the duration of the distributed application would not be that high to allow for a lot of movement of the nodes, and for most applications, they can be assumed to be stationary. A more realistic topology would be where nodes are distributed across whole permissible range (0 - 20 dB as shown in Fig. 2.11) of attenuations. Series "Combined" illustrates one such topology. The assigned attenuation values are as shown in table 2.2

Table 2.2: Attenuation settings

Node	Attenuation (dB)
Node 1	0
Node 2	5
Node 3	10
Node 4	15
Node 5	16
Node 6	17
Node 7	18

As shown in Fig. 2.11, with this topology, the BS tries to maintain same throughput for all the nodes, i.e., it averages out the throughput so as to share the channel in fair manner.

2.1.2 Hadoop

Hadoop is an open source implementation of Google MapReduce [7] and Google File System (GFS) [2]. Hadoop is created by Doug Cutting (the creator of Apache Lucene a widely used text search library) while developing an open source web search engine called Apache Nutch.

Hadoop eco-system is based on Google Cluster Architecture [14]. The architecture differs from traditional High Performance Computing in the sense that, the performance does not depend on high-end server hardware. The reliability of the system is a embedded into the software rather than high capacity server hardware. This helps us to build a high performance computing cluster at much cheaper costs.

The goal of Hadoop is to process large-scale data in a clustered environment. Hence, it is imperative that, the computing nodes in the cluster have fast access to the data. To satisfy this requirement, it is required to have high node-to-node bandwidth, which is not the case with distributed mobile cluster. Hence, design of network architecture is one of the fundamental concerns in achieving high performance.

Because Hadoop relies on commodity hardware as its underlying substrate, it brings down the total cost of deployment considerably. This is the architecture used for most of Hadoop deployments in practice. But of course, there are limitations that arise due to this kind of architecture. Commodity hardware tends to show high possibility of failures at various levels: Node, Network. etc. Hadoop ecosystem makes sure that the cluster and all its running jobs endure, and terminate past any localized failure. Drawback of using commodity network elements is: the aggregate bandwidth of the entire cluster becomes relatively low. This originates problems when performing large-scale data processing.

To overcome this limitation, Hadoop ecosystem considers network bandwidth as a scarce resource and implements mechanisms to optimize its usage as far as possible. To achieve this, the network topology is carefully modeled and studied in Hadoop.

Network Topology in Hadoop

As shown in Fig. 2.12, Hadoop represents the topology of cluster in the form of

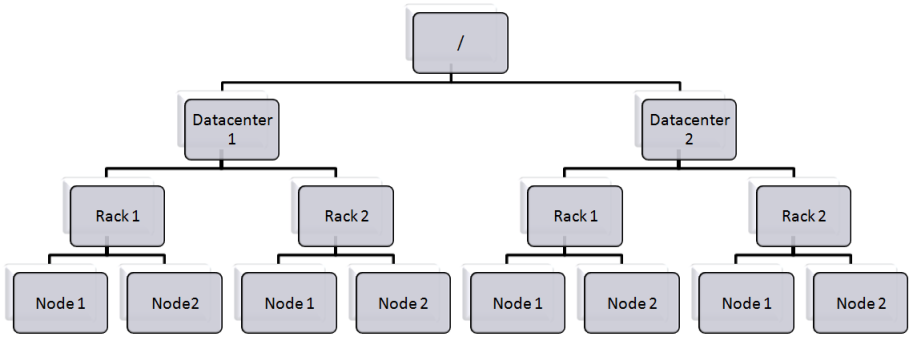


Figure 2.12: Network Topology in Hadoop

a tree and the nodes in the cluster can be viewed as leaves of the tree that reside in a rack. A rack typically consists of 30-40 nodes connected to each other by a 1 Gbps switch. It cane be noticed that, in mobile cluster, there is no notion of rack, and the interconnect is with wireless links of low, fluctuating bandwidths with variable latency.

The distance between any two nodes in this topoligy is equal to the the number of hops in between them, i.e. sum of their distances to their closest common ancestor. This definition of distance also changes in mobile cluster. In this case, the distance can be attributed to the fundamental characteristics of a wireless link: *bandwidth* and *latency*.

The Hadoop framework mainly comprises of following components:

- Hadoop Distributed File System (HDFS)
- MapReduce

Hadoop Distributed File System Architecture

HDFS is a distributed, portable and scalable rack-aware file system. It consists of a controller node - NameNode and a cluster of worker nodes - DataNodes. The files are split in blocks of 64MB and distributed across DataNodes. HDFS replicated every file on multiple nodes depending on the supplied replication factor (default: 3). Replication makes sure that the file is available irrespective of node failures. HDFS re-replicates a file if the number of replicas fall below the designted factor.

Hadoop MapReduce Architecture

The MapReduce is a programming model for distributed which essentially simplifies the job of parallel programming for the application developer so that he can focus on speeding up the deployment. All the tedious details about parallelization, data distribution, cluster management, fault tolerance, inter-process communication, are left to the underlying Hadoop framework. The programmers only job remains to model a problem in terms of Map and Reduce pattern.

MapReduce programming model essentially consists of three phases:

Map: This phase performs a parallel task on given data. The input data on which the processing is supposed to be done, is split across multiple servers across the cluster. On all such servers that already own the data, Map tasks get scheduled. The input of Map function needs to be in form of key-value pairs. RecordReader does this job of converting input file into key-value pairs. e.g. if the input file is a text file, then the RecordReader reads the file line-by-line, assigns line number to key and actual line to value. This key-value pair is then processed in Map function. This is explained in figure below. The exact task to be executed inside Map function needs to be provided by the programmer. Because Map phase executes the same task on all input splits, the programmer needs to make sure the functioning of Map function does not depend on input data. Map phase is completely parallel, i.e. all input splits are processed in parallel, hence it provides highest speedup. The output of Map phase is written to a temporary file in terms of key-value pairs. With Hadoop framework, the only responsibility that the programmer has to carry out is writing these Map and Reduce functions. Hadoop framework, using its cluster management services, decide which server to put the task on to. This makes programmers life much easier.

Shuffle: After Map phase generates its output in terms of key-value pairs, shuffle-and-sort phase sorts all key-value pairs generated by all map tasks, based on keys. After the keys are sorted, similar keys are put in same bucket and forwarded to Reduce functions on the nodes that run reduce tasks. As it can be observed from the figure below, a lot of data transfer happens during this phase. If the application is not carefully written, shuffle phase might end up exchanging all the input data across machines. When the available bandwidth is low, such as our case, this phase can become

a bottleneck. This phase can potentially limit the type of MapReduce application one can run on distributed mobile cloud. If the application requires a lot of data transfer during this phase, or to put it in other words, if Map output is large in size, there will be huge data transfer among nodes of the cluster, slowing down entire job. It is clear that, to optimize the overall performance, this is the phase which shows lot of potential and should be addressed. This phase is written by Hadoop framework and application programmer has no control over it.

Reduce: This is the last phase of MapReduce programming model. In this phase, all the key-value pairs are combined to get the final result. Similar to Map function, Reduce function is written by the application programmer. At the end of reduce phase, the output (key-value pairs) is written back to HDFS in form of required OutputFormat.

2.2 Related Work

2.2.1 Hyrax: Cloud Computing on Mobile Devices using MapReduce

Hyrax [15] is a Mobile Computing platform derived from Hadoop for Android devices. Using Hyrax, the client applications can upload their data into the heterogeneous cluster of mobile devices and servers, and execute a computation on them. Hyrax provides scalability with number of devices and tolerates node departure. Hyrax provides the abstraction of distributed computing platform, where the underlying physical nature of the cluster is transparent to the client applications. Hyrax uses Hadoop as a baseline platform in order to avoid "re-inventing the wheel". However, there are certain disadvantages of using Hadoop directly on mobile devices as described below:

- Hadoop is designed to be non-conservative in CPU and memory usage. Being optimized for I/O bound jobs, Hadoop performs well for the jobs in which read/write are the most time-consuming operations. Following qualities of the Hadoop codebase exemplify this aspect:
  - Heavy use of inheritance and interfaces lead to computational overheads due to lookups.

  - Hadoop assumes that the available memory buffer is of at least 100MB, which does not hold for mobile devices.
  - Extensive use of XML: Parsing XML is an extremely expensive task.
  - Use of servlets: Hadoop makes extensive use of servlets even though more economical options, such as HTML are available.
  - Use of JSPs: Hadoop uses JSPs that require dynamic compilation. This approach is found to be inefficient for mobile devices.

- Hadoop lacks the ability to harmonize with the slow and varying network connections, which is a typical case for cellular networks.

- HDFS lacks ability to support disconnected operation, which is a common case in mobile environment.

Some of these issues with Hadoop are addressed by Hyrax in order to port it to android devices. In Hyrax, NameNode and JobTracker are not ported to mobile devices, and should run on traditional servers. Following contributions were made in Hyrax in order to overcome Android and Hadoop obstacles:

- Tackling Anrdoid obstacles: Many of the dependencies for Hadoop were rejected by Dalvik JVM during its consistency check. Hence the source code for the libraries was downloaded and recompiled. Also, Android can only run .apk files and not .java files. Hence an application was developed to invoke DataNode and TaskTracker processes as different server processes. To add to it, the compability issue of different UNIX shells was resolved. Additionally, the job class files were packed as a worker application in Hyrax in order to be able to execute arbitrary jobs on TaskTrackers.

- Tacking Hadoop obstacles: As mentioned earlier, Hadoop allocates memory buffers of 10 to 100MB. Andoid only supports upto 16MB of heap size. Hence the buffer was reduced to 1MB and io.sort.record.percent parameter was adjusted. In addition to it, Hadoop default timeout was increased using dfs.socket.timeout parameter. Also, to overcome the parsing problem, XML configuration files were

replaced with properties files. The block size of 8MB was used in order to avoid overflowing of heap

As can be seen, Hyrax is an extension of Hadoop that is portable on mobile devices. It is more focused on compatibility with the mobile platform than performance on wireless network. Our thesis precisely concentrates on improving those aspects of Hadoop. Some of our design decisions, such as, HDFS block size, timeout values and sort record percentange are derived from Hyrax.

2.2.2 CODA: A highly available file system for a distributed workstation environment

Coda [16] [17] is a large-scale distributed file system for a computing cluster of Unix workstations. It is an extenstion of *Andrew File System (AFS)* [18] in the sense that, apart from the traditional features of distributed file systems (e.g. ACID properties), Coda provides robustness against network and server failures. It incorporates two mechanisms, namely - *Server Replication* and *Disconnected Operation* so as to achieve this resilience.

- Server Replication: In Coda, the replication is performed in units of *volume*. A volume is a collection of files and directories placed on one server that forms a partial subtree of the shared namespace. This volume is replicated to a set of servers, called as *Volume Storage Group* (VSG). At the time when such a volume is created, its replication factor and target servers are specified. Coda utilizes a variant of *read-one, write-all* approach to maintain consistency. Once a file has been closed after modification, the mutations are transferred to all the members of the VSG in parallel. The client can connect to any server out of the VSG for update. It keeps a record of the subset **s** of the servers from VSG it could connect to in the past. The client also updates all the **s** serves when there is a mutation in a local copy. When opening a file, the client also checks if it has the most recent copy. This is known as *First Class Copy*.

- Disconnected Operation: Coda assumes that the clients can be arbitrarily turned

off or disconnected. A client is said to be disconnected when it cannot connect to any member of the VSG. In such a scenario, the clients continue their normal operation of serving and updating the file, while attempting re-connect with the server in the background. Therefore, the user is allowed to be completely oblivious of disconnected operation, unless there is a cache miss. When the connection is re-established, the conflicts, if any, are resolved using reintegration.

It can be observed that, Coda concentrates on providing highly available and scalable distributed file system with the help of strategies such as, Server Replication and Disconnected Operation. Although HDFS supports namespace replication using SecondaryNameNode, it does not provide resilience for disconnected operation. It can be noted that, Coda does not take into account, any information regarding the link quality of the node that could prevent disconnected operation altogether. As base stations have prior knowledge of link quality and movement of a node, it can be used as a feedback to allow for the NameNode to make informed decision for replica placement, which is one of the contributions of this thesis.

2.2.3 D2MCE: Design of Dynamic Distributed Mobile Computing Environment

D2MCE [19] is a distributed mobile computing platform which facilitates dynamic cluster. The computing resources in the cluster can join and leave at any point without affecting the overall state of the cluster. A group of mobile nodes in D2MCE is called as *dynamic computing group*. Instead of message passingm D2MCE supports a distributed memory abstraction to enable a means of communication for the nodes. A multithreaded programming model is used for ease of programming. It uses Release Consistency model to solve the consistency issue of distributed shared memory system.

In D2MCE, multiple mobile devices connected through a wireless network form a *computing group*. A coordinator is selected among these nodes that serves as a master node, and other nodes join its group as they arrive.

D2MCE is implemented as a library on top of Unix based operating systems. It

provides a set of functions that help it achieve the dynamic environment. d2mce_init() and d2mce_exit() are the functions used to initialize and finalize the use of D2MCE library in a multithreaded application. The functions d2mce_join() and d2mce_leave() are used by the nodes for dynamically joining and leaving the cluster. Once a node joins the cluster, it shares some of the load of other nodes. When a node leaves the cluster, the cleanup is performed in d2mce_leave() function. There are many more functions for shared memory allocation, deallocation and synchronization, etc. that together form the D2MCE computing model.

The purpose of D2MCE is mainly to allow a dynamic cluster and it does not take into account the inherent issues of the cellular network such as variable bandwidth and intermittent connectivity. Also, HDFS allows for joining and leaving of the nodes, but does not support disconnected operation. Our extensions to HDFS allow it to take into account the link quality and network topology before selection any node for data placement.

Chapter 3

Design and Implementation

3.1 Design and Implemention

For several years, Internet technology has successfully brought services to our homes and greatly simplified our lives. Modern Internet is progressing towards providing services-on-the-go, like email, web surfing etc. for portable devices. Combining this strength of cellular network with sensor capabilities of current mobile devices can give birth to a whole new set of useful applications. e.g. In a vehicular network, using positioning sensors (GPS) and accelerometers, a rough prediction about traffic conditions can be made by processing the readings collected from all the devices in that area. Similar approach can be used to avoid accidents in automated vehicles. Such real-time applications entail superior end-to-end system performance and slight delay in response can result in severe loss, making it crucial for the application developers to focus on response time. This puts a lot of pressure on the back-end systems, which are predominantly centralized with limited capabilities. To address this problem, modern applications like Google Search Engine, Facebook, Amazon etc., are advancing towards distributed system architecture.

DMC takes this concept one step ahead, and uses mobile devices as the computing backbone. As explained in the previous sections, mobile devices have grown tremendously in their computation and communication capacities. To add to it, worldwide penetration of mobile devices is increasing exponentially. Past research on human mobility has proven that, most of the mobile devices do not demonstrate high mobility in reality, and are stationary for maximum time of the day. Recent data suggests that, human travel pattern is consistent with Lévy Flight [20]: Humans tend to travel short distances for most of the times (between home and work), with occasional longer

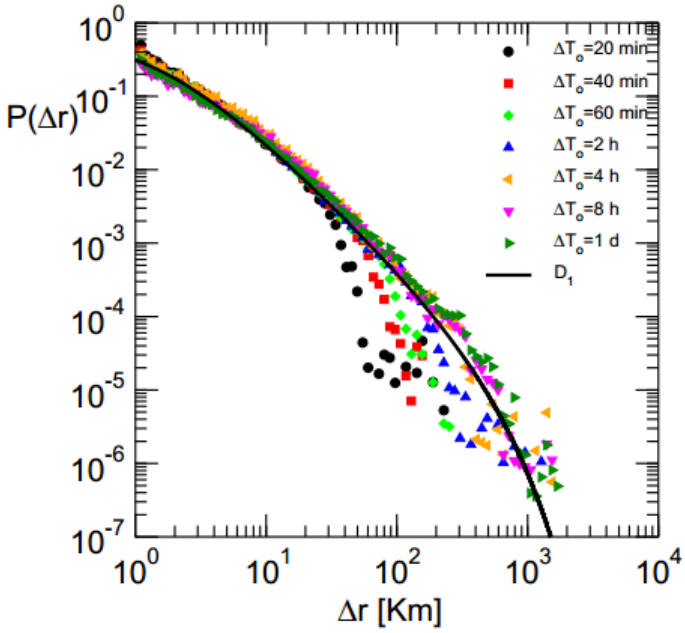


Figure 3.1: Displacement distribution P(Δ r) for xed inter event times Δ T0 for dataset D1

trips [21]. The study in the cited article is based on the dataset collected by a European mobile phone carrier for billing and operational purposes. The dataset consists of 100,000 records selected from active mobile devices, making or receiving calls. Total of 16,364,308 positions were captured. The distribution of displacement of all the users can be approximated by Eq. 3.1.

$$P(\Delta r) = (\Delta r + \Delta r_0)^{-\beta} exp(\frac{-\Delta r}{\kappa}) \tag{3.1}$$

Fig. 3.1 demonstrates the distribution of displacement. As shown in the figure, the probability of displacement is inversely proportional to the amount displacement. This can be ascribedi to the fact that most people working in offices are stationary for most time (8 hours). The service provider can monitor movement of devices using the Call Detail Records (CDR) and create a profile. The service provider can also monitor the activity of the devices, e.g. frequency of data access during the day etc. This information can be used in Hadoop's node selection process. e.g. the devices that are idle and expected to be stationary for next few hours can be given priority over the

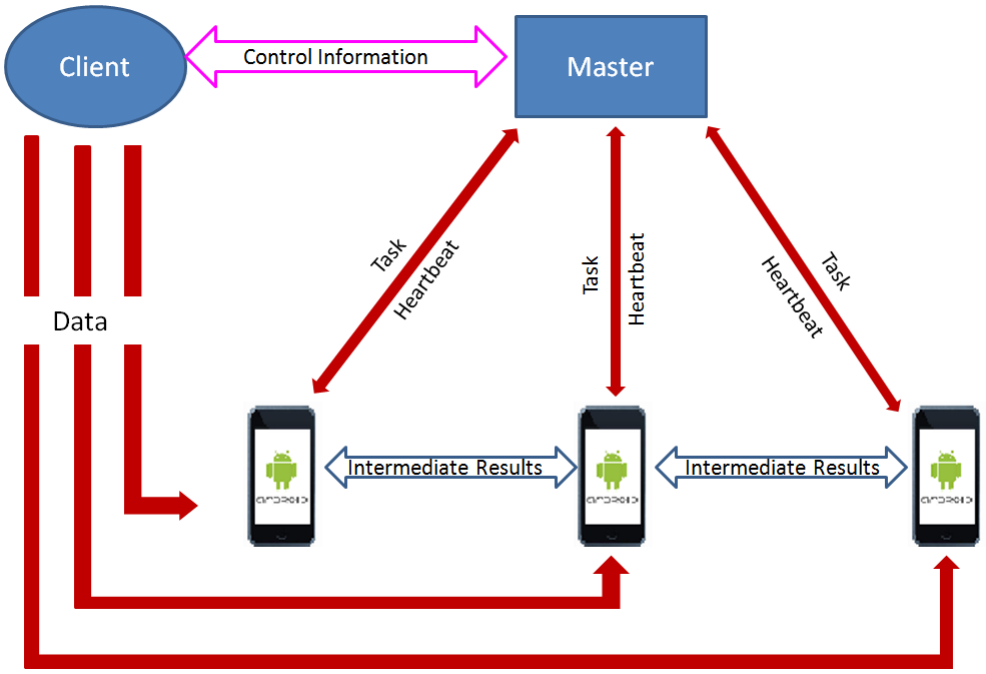


Figure 3.2: Architecture

moving and busy ones.

As can be seen, mobile devices can serve as a great backbone for distributed computing. In this thesis, we have developed a plaform for DMC using Hadoop. After analyzing the functioning of Hadoop ecosystem, we realized that the plaform, as is, cannot work efficiently in mobile environment. Moreover, it is a heavy platform and is not readily compatible with mobile platform. Hyrax [15] was developed with the objective of building a lighter Hadoop more attuned with the mobile platform. In this thesis, our focus is on improving the communication aspect of Hadoop in lieu of cellular network.

3.1.1 Architecture

DMC comprises of a cluster of mobile devices, interacting with each other to accomplish a complex task. The architecture of DMC is shown in Fig. 3.2.

This platform is designed with following assumptions in consideration.

- The cluster can be heterogenous (some nodes can be wired). For the purpose of our experiments, we have only used wireless nodes.

- The *Master* (NameNode and JobTracker) communicates with the base station controller over a wired link.
- The *Client* is not a wireless device, and can communicate with the base station controller and master node over a wired link.

Each node in the system performs a specific role assigned to it by the Hadoop component running on it.

- Client node runs Hadoop client (DFSClient and JobClient), which uploads data into the cluster and inserts information related to the associated MapReduce task.
- Controller node runs the Hadoop master components - JobTracker, NameNode and SecondaryNameNode. The role of the controller is only to store and supply control and metadata information and does not participate in any data/task related activity. Additionally, the controller communicates with the WiMAX base station controller in order to gather SNR information.
- Mobile devices in the cluster act as *Slaves* and run Hadoop's DataNode and TaskTracker components.

The client uploads the file into HDFS using DFSClient. The NameNode stores the mapping of "File ⇒ Blocks", and "Blocks ⇒ DataNodes". Every block is replicated multiple times, depending on the replication factor (default: 3). Once the file is uploaded, JobClient initiates the MapReduce job. The JobTracker requests NameNode for the list of DataNodes storing associated data, and deploys the task on them. Once the task is completed, the TaskTrackers store the output in HDFS. The client can then download this output from HDFS with the help of DFSClient.

It should be noted that, the terms *Upload* and *Download* are used in terms of HDFS. In terms of FS shell commands, "Upload to HDFS ⇒ copyFromLocal" and "Download from HDFS ⇒ copyToLocal". These terms should not be confused with *Uplink* and *Downlink* in the context of WiMAX. Download from HDFS associates to Uplink from node to base station, whereas Upload to HDFS associates to Downlink from base station to nodes.

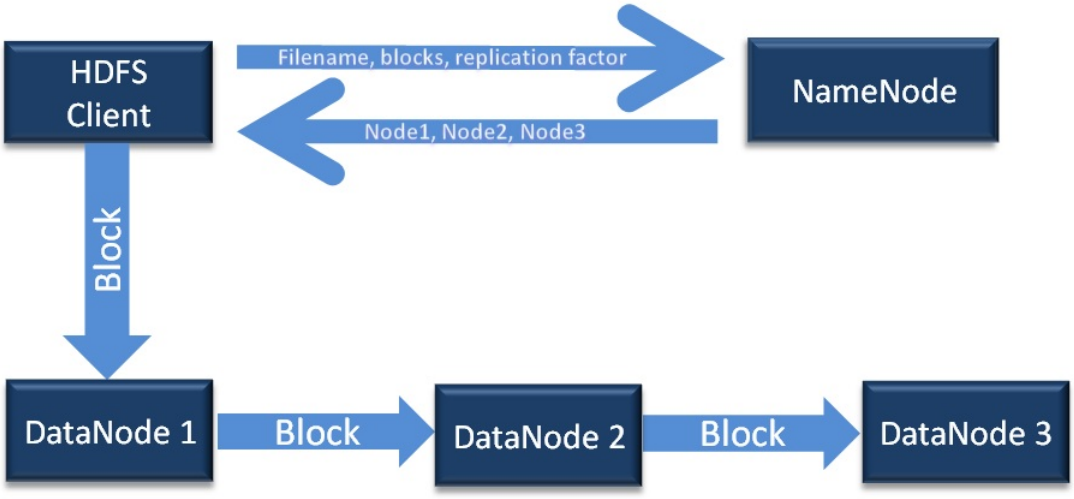


Figure 3.3: Pipelining in Hadoop

The following section explains the design and implementation of DMC platform using modified Hadoop.

3.1.2 Design and implementation of Unpipelined HDFS

In current implementation, when a client wishes to create a file in HDFS, for each block of that file (depending on file size) it requests NameNode for a list of potential DataNodes (= replication factor). The NameNode updates its namespace and provides client with an array of nodes, ordered by their distance from client. This list of DataNodes is called as *Pipeline* in HDFS. In pipelining, as shown in Fig. 3.3, the client sends the file block to the first node in the list, and requests it to forward it to the next Datanode. When first Datanode receives this block, it copies it to its own disk and then forwards to the next, at the same time, requesting it to forward to the third node, and so on. This flow of data from wireless network point of view is illustrated in Fig. 3.4

As depicted in Fig. 3.4, the client sends the block to DataNode 1, which in turn forwards it to DataNode 2, and so on. In wireless environment (assuming single base station), this implies that 5 copies of same data would be transferred on the same channel at the same time. Furthermore, two of those copies would be transferred on uplink datapath and would bottleneck the whole pipeline, causing overall throughput

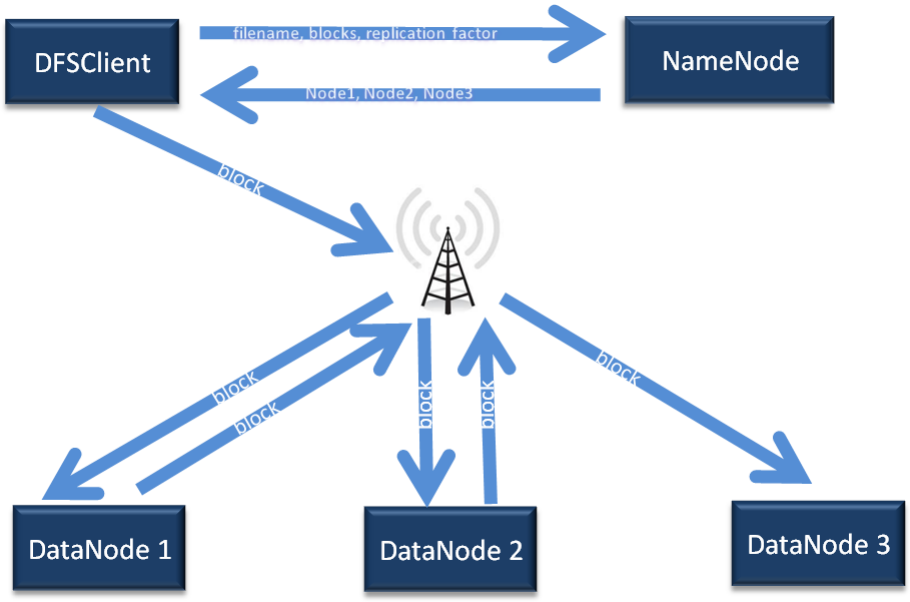


Figure 3.4: Pipelining in Hadoop - Wireless Network View

to reduce to mere 0.5 Mbps (1 Mbps/2 - as observed in our setup). This is a major drawback of current Hadoop implementation that needs attention. One way to solve this problem is to use broadcast or reliable multicast. The problem with both these approaches is that, the transmitter automatically switches to lower modulation schemes when broadcast or multicast is detected. Effectively, the throughput will again be as low as 1 Mbps. An alternative technique to solve this problem is to remove the bottleneck by omitting the uplink datapath. Assuming the client to be a wired node (wired connection to the controller) and not part of the cluster, instead of only one, the client can transfer the block to all three datanodes on three seperate downlink streams simultaneously. This results in data throughput of 6 Mbps (18 Mbps/3 - as observed in our setup), which is much higher than earlier case.

In current implementation, DFSClient opens a TCP connection to the writer. It generates a data packet that includes the block to be sent and location information of the next nodes in the pipeline. The DFSClient also creates a stream to track the acknowledgements received from all these nodes. The last node in the pipeline sends its acknowledgement to the second to last node, and that node, in turn, forwards it to the node previous to it, along with its own acknowledgement, and so on. Hence, the

data flows from first node to last node using DFSOutputStream, whereas the acknowledgements flow from last node to the first using DFSInputStream.

To implement Unpipelined HDFS (implemented in DFSClient), the message forwarding on both, OutputStream and InputStream is omitted. Instead, DFSClient opens three (= replication factor) separate connections to all the DataNodes. The packet that is sent to all these DataNodes does not contains any information about the remaining nodes in the pipeline and includes only the data block. Similar logic extends to the InputStream of acknowledgements. The DataNodes in the pipeline now send their acknowledgements directly to the DFSClient on individual streams rather than forwarding to the previous node.

DFSClient being a wired node, all the downstream data from DFSClient is first sent from node to base station, and then from base station to mobile nodes on downlink datapath. The downlink datapath has much higher bandwidth than uplink. In contrast, the acknowledgements from mobile nodes to client are sent on uplink data path from mobile node to base station, and then on wired link from base station to client. The size of the acknowledgement message is much smaller than data. Therefore, in Unpipelined HDFS, majority of the data is transferred on downlink datpath, whereas all the control information is sent on the uplink datapath. This strategy accounts for most of the gain in the system.

3.1.3 Design and implementation of Base Station aware HDFS (BAHDFS)

HDFS utilizes the principle of rack-awareness for replica placement. It places first replica on a server that is closest to the client. For second replica, it chooses a different rack, to avoid data loss due to rack failure. For third replica, it uses the same rack as the second replica, but selects a different server. When it comes to a cluster of wireless devices, the visualization of rack starts to fade. Hence, for a wireless network, different approach needs to be used, where we define Base Stations as Racks, and Mobile Devices as Servers. The system is designed in such a way that HDFS selects different Base Stations for each replica. By doing so, we avoid the contention between different nodes under same Base Station. The approach is explained in Fig. 3.5. This

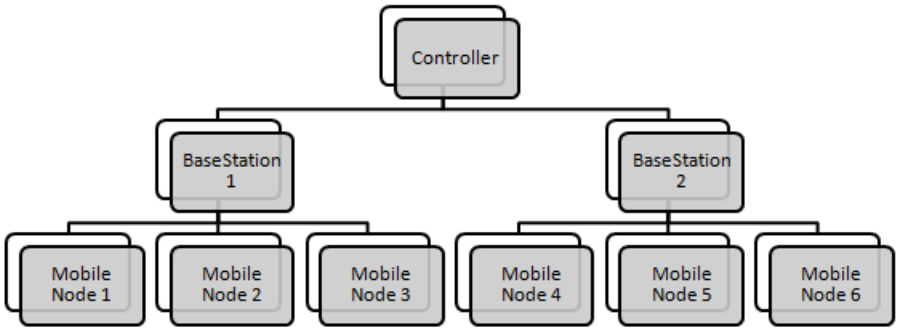


Figure 3.5: Rack Awareness In Wireless Network

strategy also improves the performance of MapReduce. Due to Hadoop's principle of data-locality, the Jobtracker brings task closer to the data. As with BAH, the data is located on different base stations, the tasks (In MapReduce, the tasks are data-parallel) are isolated from each other. Hence, using different base stations we can avoid overloading any single base station.

BAHDFS is implemeted in NameNode as a part of ReplicationTargetChooser. The default implementation of ReplicationTargetChooser is as follows:

- For first replica of a new block, the TargetChooser calls chooseLocalNode, which returns a node local to the client, to be selected as *Writer*. If the client is not part of the cluster, chooseLocalNode returns any leaf node at random from the NetworkToplogy.

- For the second replica, the TargetChooser calls chooseRemoteRack, that returns a node from a different rack from the first node. If no remote rack is available, this function returns any leaf node at random from NetworkTopology.

- For the third replica, the TargetChooser sees if the two previously selected nodes are on the same rack. If yes, it calls chooseRemoteRack function, that returns a node from a different rack than that of first and second node. If no, it calls chooseLocalRack that returns a node from the rack of first node or the the second node.

To implement HABDFS, a topology script file is provided, that stipulates instructions to treat every base station as a new rack. This file is used by HDFS to generate NetworkTopology. The file name is specified using parameter *topology.script.file.name* in hdfs-site.xml configuration file. Also, the TargetChooser function of ReplicationTargetChooser is modified so that it calls chooseRemoteRack for every replica.

BAHDFS helps in selecting a new base station for every replica. Therefore, it avoids sharing among the nodes under same base station, thus improving overall throughput of the system.

3.1.4 Design and implementation of SNR and Load Based Node Selection system (SLBNS)

MapReduce programming paradigm involves a significant amount of data transfer before (data upload), during (sharing intermediate results) and after (copying of results) the execution of task. This suggests that efficiency of the distributed task greatly depends on the throughput, and hence, on link quality. In a mobile environment, devices are scattered across the coverage area of the base station, resulting in a topology with diverse link SNRs. Hence, for a particular task, it would help to store data on the devices with higher SNR so as to get better performance. This notion motivates the design decision of SNR and Load Based Node Selection (SLBNS). In SLBNS, when uploading a file into HDFS, the NameNode first talks to the base station controller, and retrieves SNRs for all the nodes. Then, it selects three (depending on replication factor) nodes with highest SNRs. The client then transfers the data to these nodes. Hence, files are uploaded and downloaded at maximum available data rates in lesser time. This also improves MapReduce performance as the intermediate results gets shuffled among the best nodes. Hence overall system performance is improved.

In default implementation of NameNode's ReplicationTargetChooser, the node selection process does not take into account quality of the link or current CPU usage, as it is inherently designed for wired systems only. The selection process only takes into account the distance between two nodes. In SLBNS, the ReplicationTargetChooser maintains a hash map, that stores the mapping of Node $\Rightarrow$ SNR. To gather the SNR

related information, NameNode communicates with the reporting service running on base station. The chooseTarget method then orders this map by SNR and selects the nodes with highest SNRs. In SLBNS, the DataNode is also modified to send current CPU usage infomation to NameNode by piggybacking it on heartbeat messages. The NameNode constantly monitors this information, and elects a node only if its current CPU usage is within the acceptable limit. To avoid repeated selection of high-SNR nodes, after every replica placement, the *writer* node is added to the list of ExcludedNodes. Hence, the array of nodes rotates by one node after every replica.

To summarize, SLBNS algorithm assists NameNode in selecting nodes based on their link quality and CPU utilization in wireless environment and slight improvement in performance is achieved by avoiding selecting slower nodes.

Chapter 4

Experimental Evaluation of Hadoop with WiMAX

This chapter describes the experimental evaluation of Hadoop with WiMAX. The results are gathered for baseline and modified Hadoop.

4.1 Experimental Setup

This section explains the setup used in most of the experiments described in this chapter.

- Cluster: The cluster consists of eight nodes from sandbox and five nodes from outdoor setup. Both support GENI WiMAX base station deployments, and have the essential hardware (Intel 6520 WiMAX card) to support WiMAX. The topology of sandbox is explained previously in Chapter 2 in Fig. 2.1 and can be referred at [10]. The outdoor setup has fixed nodes as well as some mobile nodes. For the purpose of this experiment,we have used the fixed nodes as they were found to be more reliable. The fixed nodes of outdoor setup are at five different locations in ORBIT/WINLAB facility, and at two locations in the CoRE building, at Busch campus, Piscataway, NJ. Each of these locations have two ORBIT radio nodes. For the experiment purposes, five nodes out of these twelve nodes are used, and all these nodes reside in the WINLAB facility in North Brunswick, NJ. The topology of outdoor setup is shown in Fig. 4.1. More information about outdoor setup can be found at [22]. Due to their fixed location, outdoor nodes have fixed SNR, and hence, constant throughput i.e., the throughput cannot be varied using an attenuator matrix, as in case of sandbox. The topology of outdoor setup is depicted in table 4.3. The topology of sandbox can be varied using the attenuator matrix, as explained in Fig. 2.2 in Chapter 2.

- Block Size: The HDFS block size (dfs.block.size) used for all the experiments in this thesis is decreased from default of 64MB to 8MB, and is derived from the design of Hyrax.

- Replication Factor: The replication factor of three is used in all the experiments. In ideal scenario, the replication factor should be calculated based on the cluster size and frequency of node failures. The cluster size in our test setup is too small to use higher replication factor than three.

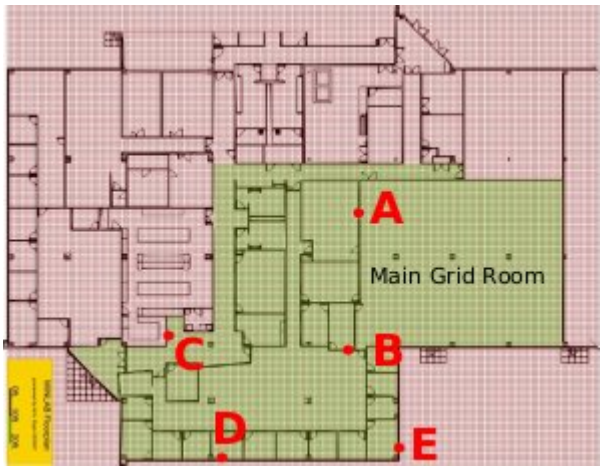


Figure 4.1: Topology of outdoor

4.2 Effect of individual design strategies on performance

This section explains the effect of each design strategy, individually, i.e. the effect on performance when only one of the stategies is implemented.

4.2.1 Effect of SNR Based Node Selection

For this experiment, sandbox was used with the settings of the attenuator matrix as shown in table 4.1. Node 5 was omitted for reliability purposes. The whole range of attenuation could not be covered due to the same reason.

In this experiment, performance of HDFS are tested with SLBNS strategy. Files of different sizes (multiples of HDFSblock sizel) are uploaded into HDFS, and the duration of upload is measured in seconds. Replication factor of 3 is assumed for the purpose of

Table 4.1: Attenuator settings for sandbox

Node	Attenuation
Node 1	0
Node 2	2
Node 3	6
Node 4	8
Node 6	11
Node 7	13
Node 8	15

this experiment. DataNode1, DataNode2 and DataNode3 denote the nodes selected by HDFS for replica placement. The number of set of datanodes depends on number of blocks into which the file is divided (e.g. for a file of 32 MB size, four sets of datanodes are required to store replicas of four 8 MB blocks). The results are tabulated in 4.2

Table 4.2: Effect of SNR Based Node Selection Strategy

File Size (MB)	DataNode1	DataNode2	DataNode3	Duration (sec)	Bottleneck rate
8	Node1	Node2	Node3	150.5	0.5 Mbps
16	Node1 Node2	Node2 Node3	Node3 Node4	315	0.5 Mbps
32	Node1 Node2 Node3 Node4	Node2 Node3 Node4 Node6	Node3 Node4 Node6 Node7	643	0.5 Mbps
64	Node1 Node2 Node3 Node4 Node7 Node8 Node1 Node2	Node2 Node3 Node4 Node6 Node8 Node1 Node2 Node3	Node3 Node4 Node6 Node7 Node1 Node2 Node3 Node4	1304	0.5 Mbps

As can be seen from table 4.2, all the nodes with high SNR are selected first, and then the array of nodes is rotated by adding writer node to the list of ExcludedNodes list. This rotation helps in avoiding repeated selection of high-SNR nodes. As evident from this result, there is no significant improvement in performance with this strategy alone, as the bottleneck rate is still 0.5 Mbps, as in case of baseline. Also, the range of attenuation used is very narrow (due to setup limitations) to prove any significant improvement.The effect of this strategy is more evident when used along with other

strategies.

4.2.2 Effect of Base Station Aware HDFS

As explained in the earlier section, in this strategy, multiple base stations are used, and HDFS is made aware of this topology using the principle of Rack Awareness. By doing so, HDFS tends to select as many different base stations as possible. For the purpose of out experiments, we have used two base stations of GENI WiMAX deployment (sandbox and outdoor). The topology used for sandbox is as shown in table 4.1. The topology of outdoor nodes is static, and as shown in table 4.3. The methodology used in this experiment is same as the previous experiment. Different files of various sizes are uploaded into HDFS and the duration is measured. The results are tabulated in 4.4

Table 4.3: Outdoor WiMAX Topology

Node	Data rate
Node OD1	10 Mbps
Node OD3	10 Mbps
Node OD6	8 Mbps
Node OD8	8.5 Mbps
Node OD9	5.3 Mbps

Table 4.4: Effect of Base Station Aware HDFS Strategy

File Size	DataNode1	DataNode2	DataNode3	Duration	Bottleneck rate
8 MB	Node OD1	Node8	Node7	137.66	0.5 Mbps
16 MB	Node1 Node4	Node8 Node8	Node OD3 Node OD3	304	0.5 Mbps
32 MB	NodeOD3 Node1 Node OD3 Node OD1	Node2 Node OD3 Node3 Node7	Node7 Node OD1 Node8 Node4	628	0.5 Mbps
64 MB	Node OD9 Node OD6 Node OD1 Node2 Node OD3 Node8 Node OD8 Node2	Node7 Node4 Node7 Node OD9 Node1 Node OD1 Node1 Node OD3	Node4 Node2 Node8 Node OD6 Node4 Node OD9 Node OD8 Node OD9	1329	0.5 Mbps

As seen from results shown in table 4.4, BaseStation aware HDFS alone shows very little benefit, as the bottleneck rate is 0.5 Mbps (uplink) in this case as well. It can be noted that, the channel sharing will be minimum if number of base stations $\geq$ number of replicas, so that for every replica, a different base station can be selected. Also, it can be observed that the selected nodes are not ordered by SNR, and the selection is completely random.

4.2.3 Effect of Unpipelined HDFS

As explained in the earlier section, in this strategy, HDFS pipeline is removed. Similar experimental methodology is used to study the effect of this strategy, as in the previous experiments. For the purpose of this experiment, only one base station (sandbox) is used. The topology for sandbox is as shown in table 4.1. The file upload durations after implementation of this strategy are tabulated in table 4.5.

Table 4.5: Effect ofUnpipelined HDFS Strategy

File Size	DataNode1	DataNode2	DataNode3	Duration	Bottleneck rate
8 MB	Node4	Node7	Node8	15	5.33 Mbps
16 MB	Node3 Node8	Node8 Node1	Node1 Node6	56	5.66 Mbps
32 MB	Node3 Node7 Node3 Node3	Node6 Node3 Node6 Node2	Node1 Node2 Node8 Node8	125	5.58 Mbps
64 MB	Node8 Node2 Node2 Node3 Node6 Node3 Node6 Node2	Node1 Node1 Node7 Node7 Node2 Node7 Node4 Node3	Node4 Node6 Node3 Node2 Node3 Node4 Node8 Node1	248	5.61 Mbps

The bottleneck rates are calculated based on average of the rates of all the nodes involved. The individual data rates are calculated using Fig. 2.7. As evident from table 4.5, the bottleneck data rates are higher as compared to baseline HDFS. This can be attributed to the fact that, in Unpipelined HDFS, the DFSClient itself takes care of the replication, and uplink datapath is completely avoided. This strategy contributes

majority of the gain in the system. As can be seen, due to lack of SNR knowledge, the nodes are selected randomly by the NameNode. Combining this strategy with SLBNS can lead to even better performance. In addition to it, if different base stations are used, the channel sharing can be eliminated, and data rates of 18Mbps can be achieved.

4.3 Performance evaluation of modified Hadoop using file upload

In this section, the experiments are performed with all the three strategies in place, and compared with baseline HDFS. The files of various sizes (from 8MB to 128MB) are uploaded into HDFS and the duration is monitored. The result is shown in Fig. 4.2.

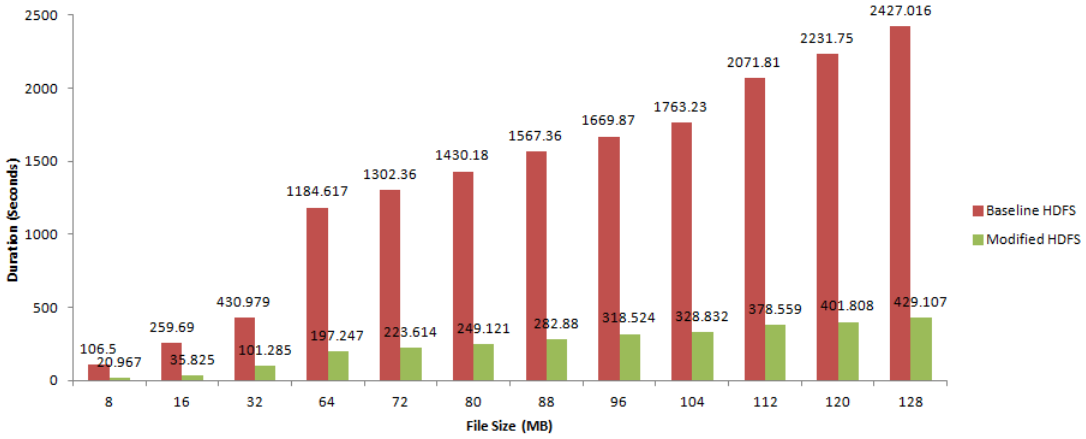
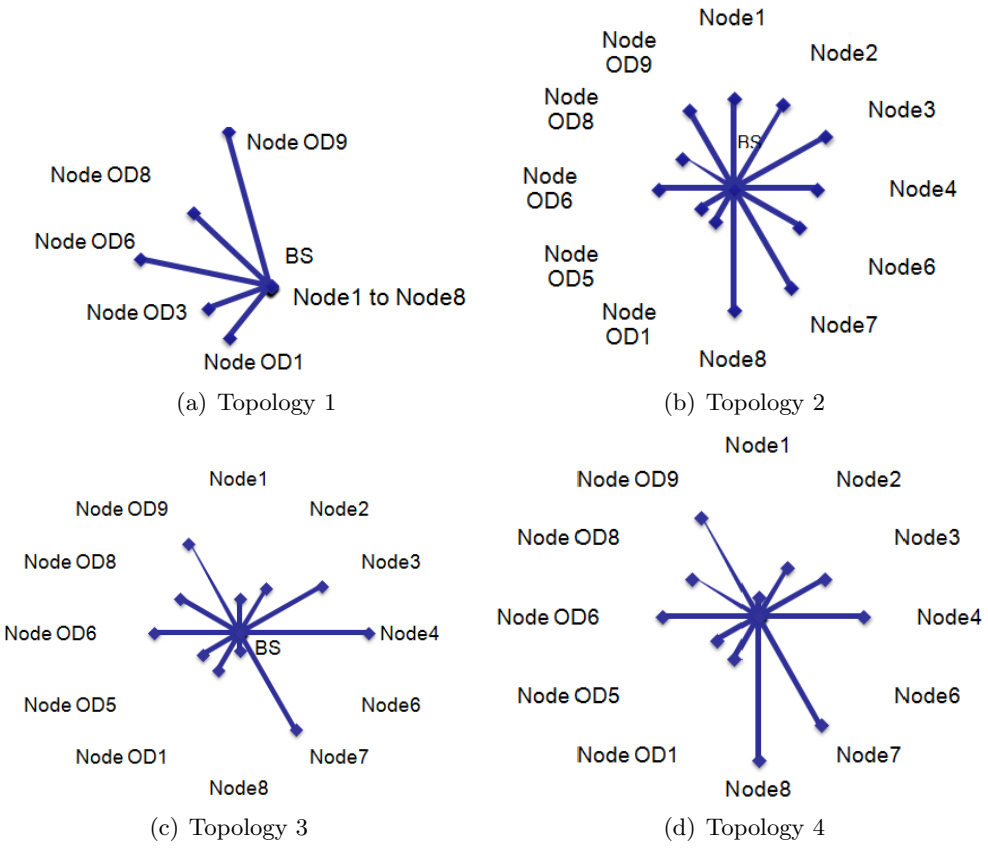


Figure 4.2: File Upload timings

As can be seen, 5x gain can be achieved with modified HDFS in file upload timings. Most of this gain can be attributed to Unpipelined HDFS. In case of pipelined HDFS, as explained in system design, for replication factor of three, there are two uplink datapaths when a single base station is used, resulting in bottleneck rate of 0.5 Mbps. Whereas, in case of unpipelined HDFS, all the data is sent on downlink datapath only. Hence the bottleneck rate can go up to 6Mbps (depending on SNR). When SLBNS is used along with unpipelined HDFS, it helps in selecting nodes with highest SNR values, which boosts bottleneck throughput to 6Mbps. When BAHDFS is combined with SLBNS and unpipelined HDFS, it helps in selecting selecting a new base station for every replica, which can result in upto 18Mbps of bottleneck throughput. it The effect of BAHDFS is not evident in these experiments due to setup limitations. At



(a) Topology 1 (b) Topology 2

(c) Topology 3 (d) Topology 4

Figure 4.3: Topologies

least three base stations would be needed in order to see any significant effect of this strategy.

4.4 Performance evaluation of modified Hadoop using Grep benchmark

In this section, performance of Hadoop is compared for baseline and modified HDFS, using the Grep benchmark (provided by Hadoop distribution). Although various other benchmarks are studied as well, Grep is analysed in more detail, as it is an example of the type of applications that benefit the most from DMC. Various different topologies are used to analyse the performance. The topologies are depicted in Fig. 4.3.

Grep benchmark is a MapReduce application that takes as input, a text file or a directory, searches for a pattern in all the input files, and counts its occurances. The mapper reads the input file line by line, and searches for the required pattern. If the

pattern is found, it outputs "pattern ⇒ 1". The reducer then simply sums up all the values and outputs final count. In this application, the ratio of output size to input size is very small. The result of Grep application is of the form "Pattern" ⇒ "Number of occurances" and its size is approximately constant for all input sizes.

The performace of Grep benchmark is analyzed for four different topologies as shown in Fig. 4.3.

The result of this experiment is depicted in Fig. 4.4. The result is collected for the three phases of this MapReduce application.

- *Upload* of the input file into HDFS
- *Execution* of the MapReduce task
- *Download* of the result from HDFS

The terms *Upload* and *Download* are used in the context of HDFS, and not base station. *Uploading* a file into HDFS entails a *downlink* traffic, whereas *downloading* a result from HDFS entails *uplink* traffic to the base station.

As can be seen in the results shown in Fig. 4.4, as the file size increases, the time required to upload that file into HDFS increases. The duration of MapReduce task is approximately contant for all scenarios due to data-parallelism of the MapReduce task. Also, the amount of data exchanged in various phases (shuffle, reduce) is very small as compared to other benchmarks. It can also be observed that, as the size of the result generated by Grep is constant, the time spent in retrieving it from HDFS is small and constant as well. Therefore, Grep benefits the most from DMC, and serves as an example of the type of applications well suited for this architecture.

As can be seen, upto 5x gain can be achieved in overall performance by using modified version of Hadoop.

4.5 Performance evaluation of modified Hadoop using WordCount and TeraSort benchmarks

In this section, the performance of modified Hadoop is evaluated for WordCount and TeraSort benchmarks. The description of both these benchmarks is as follows:

- **WordCount**: A WordCount application reads text files, and outputs the number of occurances of every word that appears in the file. The output is in form of a text file, where each line contains a word and its count in the file. In WordCount application, each mapper reads line from input text file, and outputs every word as key and 1 as value. The reducer adds the counts for every word and emits a single key-value pair (word - sum) as output. It can be noted that, the amount of data shuffled in between map and reduce phases is much larger in WordCount, as compared to Grep. In contrast with Grep, the size of the output in WordCount varies in proportion with input (the output size is constant in grep), but is still very small as compared to input.

- **TeraSort**: TeraSort benchmark consists of three MapReduce applications:

  - *TeraGen*: Teragen is a MapReduce application that generates the data for TeraSort in the required format. Being a MapReduce application, it takes more time to generate the data than uploading a file of equal size into HDFS.

  - *TeraSort*: The goal of this MapReduce application is to sort 1TB of data in parallel by distributing the data across the cluster. TeraSort operates on the data generated by TeraGen. Mapper is simply an identity function that reads key-value pairs from input and outputs them. The *Shuffle and Sort* phase then sorts this intermediate data and supplies to the reducer. The reducer then simply emits this sorted data to output. It can be noted that, the amount of data shuffled during the intermediate stages is much larger than WordCount and Grep, and is at least 4x of WordCount (assuming 1 byte character and 4 byte integer). Also, the size of the output (sorted data) is same as input.

– *TeraValidate*: TeraValidate is a MapReduce application used to validate the sorted output data. For the purpose of performance evaluation, this application is not used as there is no equivalent application in Grep and WordCount benchmarks.

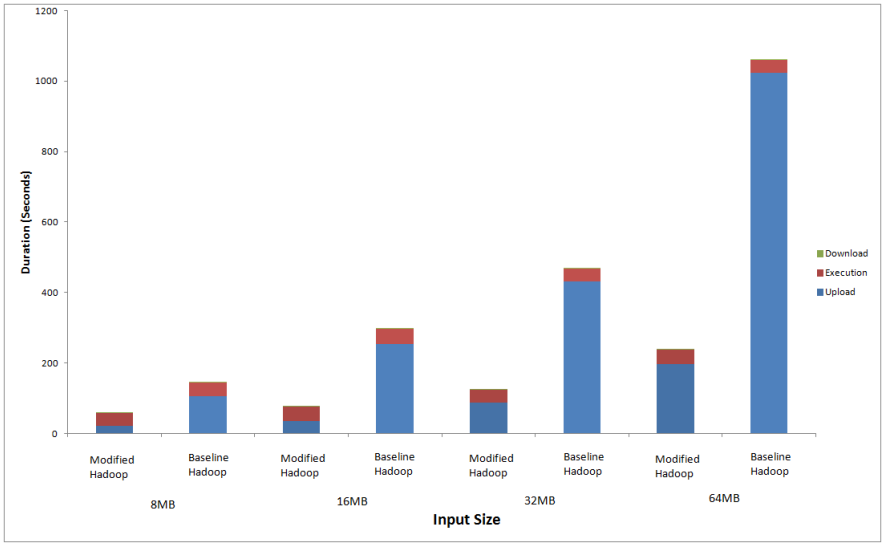
The topology used for this experiment is depicted in Fig. 4.4(a). The goal of this experiment is to use different types of benchmarks to show how some applications may be well suited for DMC, while other applications may not be. The three benchmarks (Grep, WordCount and TeraSort) are compared for different input sizes (8MB to 64MB). The result is shown in Fig. 4.5.

It can be noted that, in case of Grep, as the input size increases, only the file upload time increases, while execution and download stages complete in constant time. In Grep, as the size of intermediate output is small, the shuffle phase takes almost the same time for all input sizes. This results is constant execution time. In addition to it, the output is a constant-sized key-value tuple (10 bytes) of the form "pattern ⇒ count", hence, the time to download the result from HDFS is constant as well.

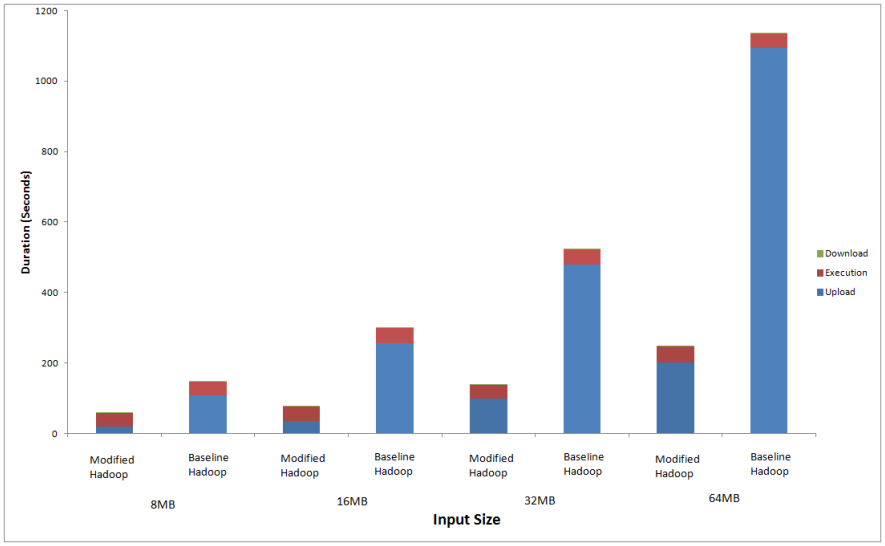
In case of WordCount, as with Grep, the time taken to upload a file into HDFS varies in proportion with the input size. As opposed to Grep, in WordCount, the size of the intermediate data also varies with input size. Hence, as the input size increases, execution time increases as well. As mentioned earlier, the output of WordCount is a text file that contains a "word ⇒ frequency" tuple for every unique word. Hence, the size of the output also varies with input. Therefore, the time taken to download the result from HDFS is not constant, and varies (but not equal) with input size.

In case of TeraSort, the amount of intermediate data that gets shuffled is large as compared to both Grep and WordCount. Also, the size of the sorted output data is same as input. It can be noted that, the downloading of result from HDFS is essentially an uplink from mobile device to base station and the bottleneck bandwidth is 1 Mbps. Hence, in comparison with previous benchmarks, the time to download the result is very high in TeraSort.

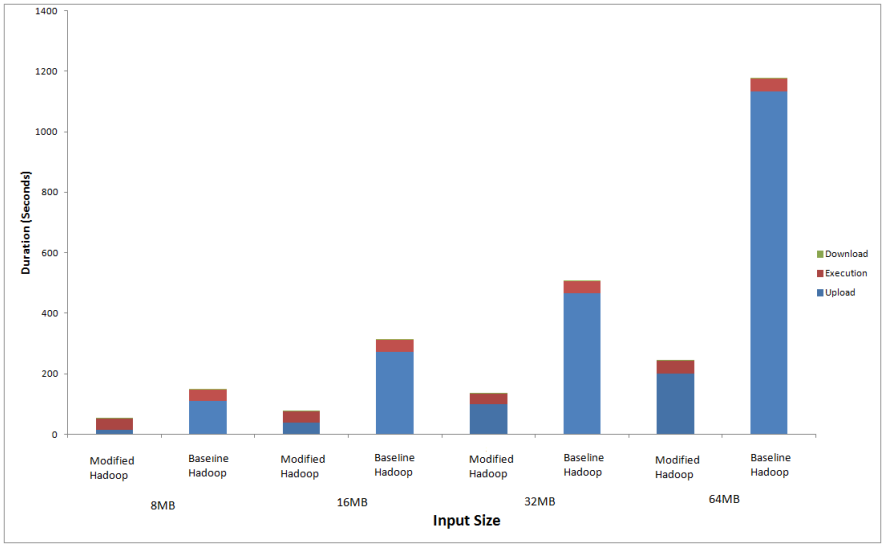
This shows that the applications of the form similar to TeraSort are not well suited for DMC. The applications where the ratio of output size to input size is small (Grep), entail minimum data transfer on the uplink (from mobile node to base station) datapath, and hence, benefit greatly from DMC. WordCount represents a class of applications that have moderate shuffled data and moderate output size, and can have more advantage of DMC over TeraSort.



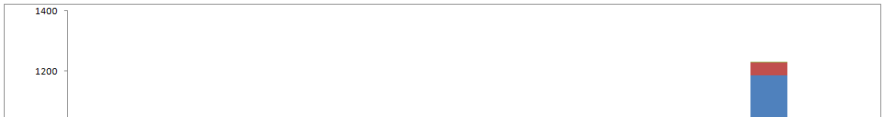
(a) Result for Topology 1



(b) Result for Topology 2



(c) Result for Topology 3



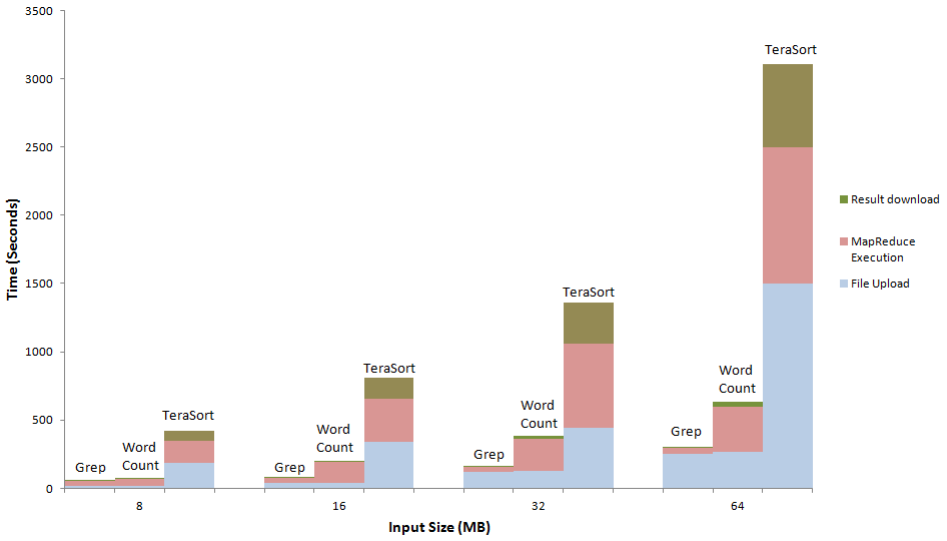


Figure 4.5: Comparison of Grep, WordCount and TeraSort benchmarks

Chapter 5

Conclusion & Future Work

5.1 Conclusion

As part of this thesis, detailed experimental study of WiMAX was performed in order to understand the behavior of cellular networks. The experiments involved various toplogies including stationary and moving nodes.

We have used existing platform - Hadoop to build an efficient infrastructure for Distributed Mobile Computing. Initial analysis of the Hadoop ecosystem lead to a conclusion that, Hadoop, as is, is not designed for wireless networks and there is a lot of scope for improvement. In this thesis, we have proposed the following three strategies to build an efficient platform for DMC using Hadoop.

- **Unpipelined HDFS:** In this strategy, the DFSClient, while copying a file into HDFS, instead of forming a pipeline, opens seperate connections to each DataNode for each replica. Hence, the uplink datapath (from mobile node to base station) is omitted, resulting in upto 5x performance improvement.

- **SNR & Load Based Node Selection:** In this strategy, the nodes are selected based on their SNR values. If selected nodes are within the limits of CPU usage, the node is selected, else it is discarded. This strategy helps in real topologies where significant portion of the mobile nodes lie in the area with lower SNR. This strategy helps in avoiding selection of such nodes in availability of better nodes.

- **Base Station Aware HDFS:** In this strategy, the NameNode is made aware of the topology of the wireless network by providing a topology script. The NameNode then prefers as many different base stations as possible in its replica

target selection process. This strategy facilitates contention minimization among mobile nodes.

The result of the experimental evaluation of modified Hadoop illustrates the effect of each strategy individually and combined. We have used standard benchmarks provided with Hadoop distribution, in order to permit comparison with other platforms. It can be noted that, the DMC architecture is only suitable for a class of applications where size of the output is small in comparison with input size. The experiments involving Grep, WordCount and TeraSort benchmarks support this proposition.

5.2 Future work

During system design of this DMC platform, there are certain aspects that are not considered in this thesis, that could have assisted in achieving better performance. Following things can be done in future in order to enhance the overall system.

- **Hadoop/OpenStack integration:** OpenStack [23] is an open source cloud operating system for building public and private clouds. OpenStack consists of projects like Nova-compute among many others, that provide the virtualization backbone for cloud servers. In Hadoop/OpenStack integration, Hadoop cluster components (NameNode, SecondaryNameNode, JobTracker, DataNode and TaskTracker) can run on the virtual machines provided by OpenStack and OpenStack can serve as "platform-as-a-service". With few OpenStack VMs on mobile devices, the inter-node communication translates to inter-vm communication on a single node. This communication happens via local socket communication, and hence is much faster than remote communication. By using virtualization, the utilization of the mobile device can be maximized.

- **Integration with Hyrax:** Hyrax is a platform based on Hadoop, designed to support Hadoop on android devices. Hyrax can be integrated with our platform and evaluated on actual android devices.

- **Experiments using bigger cluster larger attenuation range:** Some of our

strategies, SLBNS and BAHDFS are not very evident in terms of performance improvement in current setting. More experiments can be performed on a bigger cluster, with much larger range of attenuation. Due to limitations of the hardware, to cover larger range of attenuation, network emulators need to be used. Using tools like netem, the bandwidth of the wired node can be throttled (by limiting rate and enforcing packet loss) to emulate a wireless link. The impact of SLBNS strategy can be proven more evidently in a large cluster with such emulated links.

- **Intelligent Task Scheduling:** Currently, for scheduling a task, the JobTracker simply elects a node that stores the data pertaining to that task. The JobTracker can be made more intelligent by providing it with some information regarding SNR or network topology. This would greatly improve the performance of MapReduce task.

- **Sample application:** As described in Chapter 4, the Distributed Mobile Computing paradigm is only suitable for some set of applications. A sample application can be developed that fits in the DMC criteria to portray the usefulness of DMC in real life. An example of such an application could be - Distributed Image Processing. Consider a scenario where a number of images are given, and the task is to search all the images for a particular object. Different images can be processed on different nodes in parallel. The result generated by each parallel task is of the Boolean type, which indicates whether the requested object was present or absent in the image. In this appilcation, the size of the output is small in comparison with the size of the input, and hence, fits in the DMC criteria.

References

[1] Wikipedia, "Distributed computing — wikipedia, the free encyclopedia."

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, Oct. 2003.

[3] "Hadoop distributed file system."

[4] IDC, "Strong demand for smartphones and heated vendor competition characterize the worldwide mobile phone market at the end of 2012, idc says."

[5] ——, "Worldwide server market rebounds sharply in fourth quarter as demand for x86 servers and high-end systems leads the way, according to idc."

[6] Hadoop, "http://hadoop.apache.org/."

[7] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[8] "http://www.wimax.com/what-is-wimax."

[9] "Wireless information network laboratory, winlab."

[10] "http://www.orbit-lab.org/wiki/hardware/bdomains/csandboxes/dsb4."

[11] "Delay-bandwidth-product http://sandilands.info/sgordon/impact-of-bandwidth-delay-product-on-tcp-throughput."

[12] "Wimax base station scheduling algorithms."

[13] Wikipedia, "Proportionally fair — wikipedia, the free encyclopedia," 2013.

[14] L. A. Barroso, J. Dean, and U. Hölzle, "Web search for a planet: The google cluster architecture," *IEEE Micro*, vol. 23, no. 2, pp. 22–28, Mar. 2003.

[15] E. E. Marinelli, "Hyrax: Cloud computing on mobile devices using mapreduce," Master's thesis, School of Computer Science Carnegie Mellon University, September 2009.

[16] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, David, and C. Steere, "Coda: A highly available file system for a distributed workstation environment," *IEEE Transactions on Computers*, vol. 39, pp. 447–459, 1990.

[17] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," *ACM Transactions on Computer Systems*, vol. 10, pp. 3–25, 1992.

[18] J. Williams, E.H., N. Sullivan, J. Rusnak, J. Menges, D. Ogle, R. Floyd, and W. Chung, "The andrew file system on os/2 and sna," in *Communications Software, 1991, 'Communications for Distributed Applications and Systems', Proceedings of TRICOMM '91., IEEE Conference on*, 1991, pp. 181–191.

[19] W.-Y. Liang, Y.-M. Hsieh, and Z.-Y. Lyu, "Design of a dynamic distributed mobile computing environment," in *Parallel and Distributed Systems, 2007 International Conference on*, vol. 2, 2007, pp. 1–8.

[20] P. Barthelemy, J. A. Bertolotti, and D. S. Wiersma, "A levy flight for light," *Nature*, vol. 453.

[21] M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi, "Understanding individual human mobility patterns," *Nature*, vol. 453.

[22] "http://www.orbit-lab.org/wiki/hardware/bdomains/boutdoor."

[23] "http://www.openstack.org/."