ACCELERATING DATA-INTENSIVE COMPUTATIONS THROUGH DYNAMIC NETWORK TRAFFIC OPTIMIZATION

A Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Masters of Science

By

Anand Arun Daga

December, 2012

ACCELERATING DATA-INTENSIVE COMPUTATIONS THROUGH DYNAMIC NETWORK TRAFFIC OPTIMIZATION

Anand Arun Daga

APPROVED:

Dr. Jaspal Subhlok, Chairman
Dept. of Computer Science

Dr. Deniz Gurkan
College of Technology

Dr. Lennart Johnsson
Dept. of Computer Science

Dean, College of Natural Sciences and Mathematics

ACCELERATING DATA-INTENSIVE COMPUTATIONS THROUGH DYNAMIC NETWORK TRAFFIC OPTIMIZATION

An Abstract of a Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Masters of Science

By

Anand Arun Daga

December, 2012

Abstract

Hadoop has been emerging as a popular distributed framework for data intensive computing in clustered environments. The main usage has been in parallel computing problems where interconnected clusters would transfer parts of the data between individual compute nodes to accomplish one job. The clusters are usually connected with shared network infrastructure where other applications also access and transfer on the same bandwidth. Specifically, Hadoop MapReduce jobs suffer when running in parallel with other traffic in the underlying network due to their sensitivity to delay between compute phases. We propose a dynamic priority mechanism realized by OpenFlow protocol on such an infrastructure with a preferred QoS policy over all other traffic. Moreover, our proposed priority mechanism can be enhanced if additional network information on traffic in the underlying network is provided. We propose to use the emerging ALTO (Application Layer Traffic Optimization) server to provide network traffic information to Hadoop. The ALTO server will be based on the industry standard, IF-MAP (interface to metadata access points protocol), to leverage publish/subscribe capabilities and the flexible schema definitions.

Acknowledgements

I would like to thank my advisors Dr. Jaspal Subhlok, Dr. Deniz Gurkan, and Dr. Lennart Johnsson for giving me this opportunity to work on something practical and interesting, their sound advice, and their great efforts during my research at the University of Houston. I am grateful to my mentors at Infoblox, Dr. Sandhya Narayan and Stuart Bailey, for guiding me through my research during my internship. I'm thankful to my friends at UH for their emotional support and help, which made my study enjoyable and fruitful. I would like to thank the faculty in the Department of Computer Science for their assistance and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data intensive computing is now widely recognized as important to the HPC community. There is relatively little work that is public on building OpenFlow enabled clusters to support data intensive computing. The main goal of this thesis is to highlight the importance of the applicability of OpenFlow to this class of problems.

1.1 Motivation

Hadoop[20] has emerged as an important platform in the area of Big Data for data intensive computing. It provides a reliable storage and analysis system that is scalable and built from standard inexpensive hardware. The distributed storage system is called HDFS[23] and is a single, consolidated storage platform. The analysis system called MapReduce framework, exploits the distributed storage architecture of HDFS to deliver scalable, reliable parallel processing services for arbitrary algorithms[6].

Though HDFS's performance depends on disk I/O performance, MapReduce

Framework's performance depends on the compute power of the cluster, thus the data movement in Hadoop clusters is vital to the overall performance of Hadoop. Even as jobs are scheduled closer to where its data lie in the cluster, there is still so much network I/O that it often saturates the top of the rack switches, as well as switches that aggregate multiple racks. In addition, MapReduce computations are pipelined and often have "hot spots" in which the computation is lengthened due to inadequate bandwidth to some of the nodes.

Our approach to address the network issues in Hadoop is to provide an application developer or user with the ability to control their network, just as they can control how much memory to assign to a task or what files an application can write to. The ability to setup paths for data transfer as needed by an application is expected to provide improvement in Hadoop performance.

1.2 Hadoop as a Distributed Application on Cluster

The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing. The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. As shown in Figure 1.1, it is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering

a highly-availabile service on top of a cluster of computers, each of which may be prone to failures. The project includes these modules:

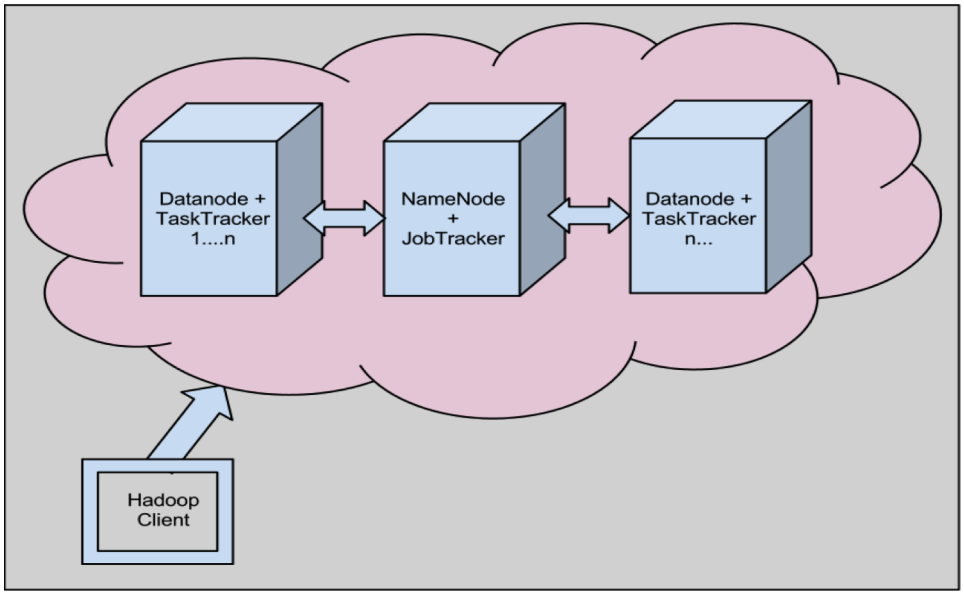

Figure 1.1: Hadoop Cluster Overview

1. **Hadoop Common:** The common utilities that support the other Hadoop modules. It provides access to the filesystems supported by Hadoop.

2. **Hadoop Distributed File System (HDFS):** A distributed file system[8][7] that provides high-throughput access to application data.

3. **Hadoop YARN:** A framework[9] for job scheduling and cluster resource management.

4. **Hadoop MapReduce:** A YARN-based system for parallel processing of large data sets.

1.3 Contribution of this Thesis

In a shared cluster, Hadoop's Shuffle traffic suffers because of the other congestion in the underlying network. We propose to create dedicated queues with preferred QoS for the Hadoop Shuffle traffic. We propose to direct the Hadoop traffic to the preferred queues by dynamically pushing flows at runtime in the OpenFlow Switches used in the underlying network. Before pushing these flows we also propose to take feedback from the Application Layer Traffic Optimization Information Base to evaluate the cheapest path in the network to forward traffic.

1.4 Thesis Outline

This thesis is divided as follows. In Chapter 2, we introduce Hadoop MapReduce Framework. We also discuss the Network Overview of a Hadoop Cluster. Applications which rely on the Hadoop Processing Framework are also mentioned here. In Chapter 3, we describe the related work done in this field of study. In Chapter 4, we put forward a much deeper explanation of our approach to accelerate Hadoop. We also explain the technologies like OpenFlow and ALTO (Application Layer Traffic Optimization), whose capabilities are being leveraged to accelerate Hadoop. In Chapter 5, we explain the Test Setup used for the experiments and the test procedures. In Chapter 6, we present the results found in our experiments. We conclude in Chapter 7 after discussing the future scope for the project.

Chapter 2

Background and Definitions

2.1 Hadoop MapReduce Framework

MapReduce [6] is a programming model where a MapReduce program can consist of two functions, map and reduce, each of which defines a mapping from one set of key-value pairs to another. MapReduce programs can be written in various languages; Java, Ruby, Python, and C++. These functions work the same for any data size, but the execution time depends on the data size and the cluster size. Increasing data size causes increased execution and increasing cluster size decreases the execution time. MapReduce programs are inherently parallel, with Map tasks running concurrently, followed by their corresponding Reduce tasks, which also run concurrently.

HDFS (Hadoop Distributed File System) is a distributed file system that stores and manages the data in a Hadoop cluster. As illustrated in Figure 2.1, there is central node called NameNode to store the meta-data of the file system and other nodes called DataNodes that store the data. Files in HDFS are split into smaller

blocks, typically 64MB, and each block is stored separately at a DataNode and replicated as per the specified replication factor to provide data durability. A HDFS client contacts the NameNode to get the location of each block, and then interacts with DataNodes responsible for the data.

Hadoop MapReduce Framework consists of two types of components that control the job execution process: a central component called the JobTracker and a number of distributed components called TaskTrackers. Based on the location of a jobs input data, the JobTracker schedules tasks to run on some TaskTrackers and coordinates their execution of the job. TaskTrackers run tasks assigned to them and send progress reports to the jobtracker.

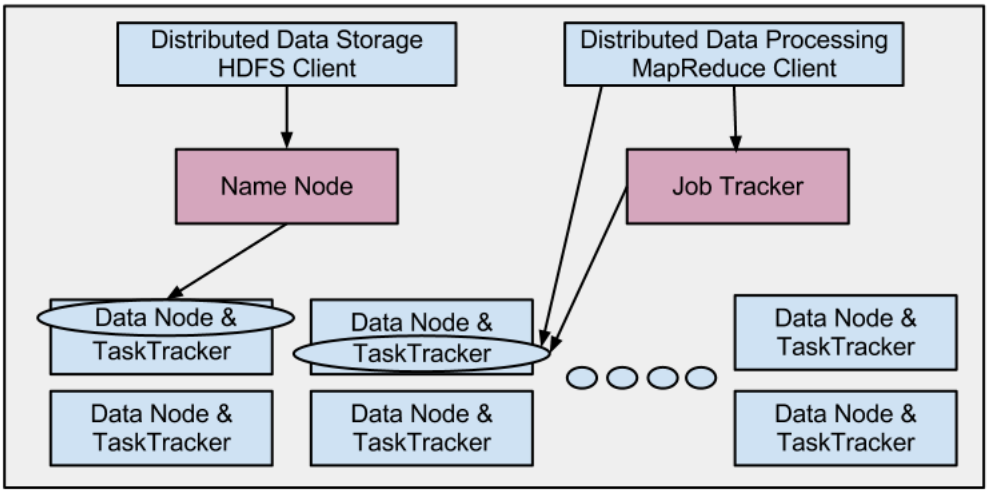

Figure 2.1: Hadoop Actors and Tasks

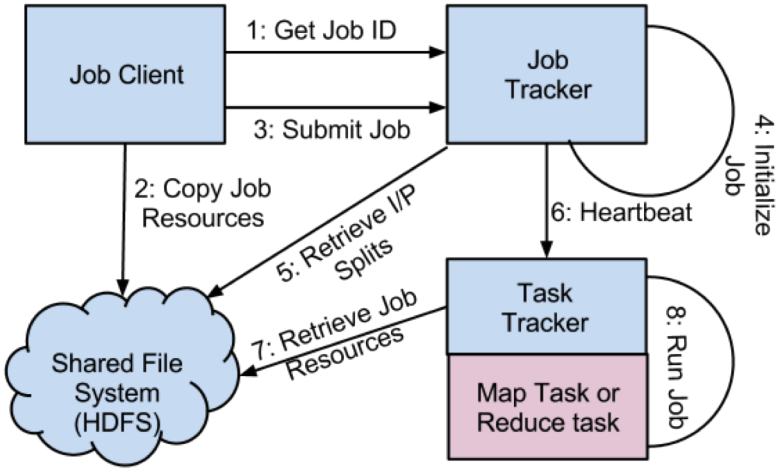

Figure 2.2: Hadoop MapReduce JobCycle

2.2 Network Overview of Hadoop

MapReduces execution model has two phases, map and reduce, both requiring moving data across nodes. Map tasks can be run in parallel and are distributed across nodes available.

As illustrated in Figure 2.3, the input datasets, that are stored in HDFS prior to beginning of the job, are divided into datasets or splits, and for each split a map task is assigned to TaskTracker on a node that is close to that split. The map computation begins when its input split is fetched by the TaskTasker, which processes the input data according to the map function provided by the programmer, generates intermediate data in the form of {key, value} tuples, and partitions them for the number of reduce tasks. In the reduce phase, the TaskTrackers assigned to do the reduce task fetch the intermediate data from the map TaskTrackers, merge the different partitions, perform reduce computation, and store the results back into HDFS.

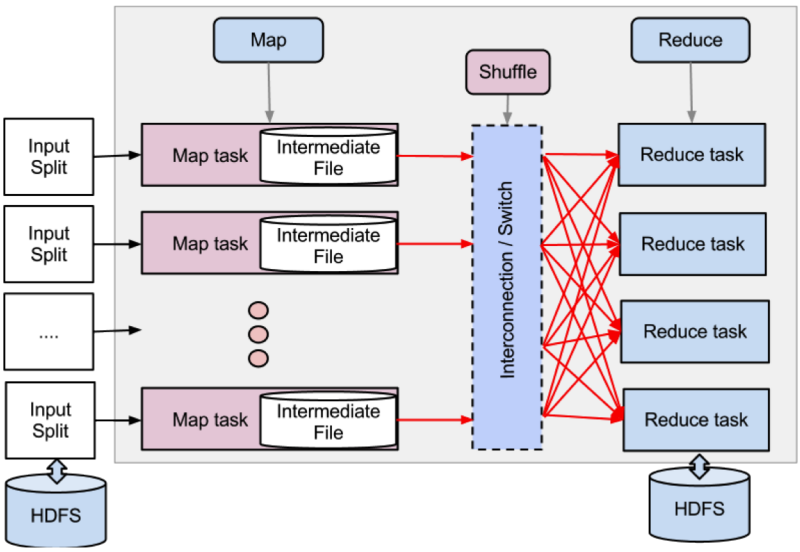

Figure 2.3: Network Communications

Network traffic in Hadoop can be separated into several categories:

1. **HDFS read and write by client**:

   This type of traffic due to data exchange occurs twice in the lifetime of one Hadoop job; once at the start of the Hadoop Job when the client writes the input payload to the HDFS, and at the end when the client reads the result written to the HDFS by the MapReduce processing framework.

2. **HDFS replication**:

   This type of traffic depends on the replication factor set by the Hadoop Cluster Manager. Replication of input payload results in reliability.

3. **Hadoop split traffic between HDFS and TaskTrackers**:

   Split traffic is generated before the Map phase starts. Split files of the input payload are delivered to the Mappers during this phase.

4. **Hadoop shuffle traffic between TaskTrackers**:

   Shuffle traffic is generated after the Map phase ends and before the Reduce phase starts. Intermediate files which are generated as the result of the processing in the Map phase are delivered to the Reducers during this phase.

5. **Hadoop results stored in HDFS**:

   This type of traffic is generated at the end of the Hadoop Job when the Reducers write the end result of the job to the HDFS.

6. **NameNode - DataNode interactions**:

   NameNode keeps a check on the health of the DataNodes by exchanging heartbeats thus generating this traffic.

7. **JobTracker - TaskTracker interactions**:

   In these interactions, Job Tracker keeps an account of the status of the task it has alloted to the Task Trackers.

If the network connecting the Hadoop cluster is able to provide timely and orderly delivery of data, and the application (Hadoop) is able to control the network based on its needs for different categories for traffic, then the application can perform more operations concurrently, and thus improve its performance.

In a Hadoop cluster, the characteristics of the network is known a-priori and can be used to the advantage of setting up a more performance network, similar to the earlier circuit-switching ATM networks. The technology that makes it possible to setup flow-paths similar to circuit switching is ONF's Openflow.

2.3 What kind of Applications need Hadoop?

The Hadoop platform was designed to solve problems which have a lot of data, perhaps a mixture of complex and structured data which don't fit nicely into tables. It is for situations where there is need to run analytics that are deep and computationally extensive, like clustering and targeting. It is used where indexing the web and examining user behavior to improve performance algorithms is needed.

Hadoop applies to a bunch of markets. In finance, to perform accurate portfolio evaluation and risk analysis, sophisticated models are derived that are hard to jam into a database engine. But Hadoop can handle it. In online retail, it is used to deliver better search answers to customers so they're more likely to buy the things shown to them. Other applications of Hadoop[28] are:
Log and/or clickstream analysis, image processing, processing of XML messages, web crawling and/or text processing, general archiving, etc.

Chapter 3

Related Work

3.1 Acceleration Techniques

Output of the Hadoop framework depends on various factors such as efficiency of the underlying distributed file system, speed with which the MapReduce processing takes place, how fast the processing framework can access the resources in the file system, how fast the intermediate file transfer happen in the underlying network, and many more. In order to accelerate Hadoop, each of the above mentioned factors have to be optimized. In this section, we discuss various approaches taken to accelerate Hadoop. Using high performance interconnects, overlapping the execution phases of Hadoop, providing early results from jobs, changing network topology based on application are some approaches presented in the research outlined below.

Hadoop processing framework involves a lot of disk access such as to read/write data to/from HDFS. In a Hadoop Cluster, traditional hard disks are used in the

underlying HDFS. But HDD's are constrained by the mechanical rotating disk resulting in poor random access performance and excessively high power consumption. Solid State Drive (SSD) has emerged as a viable alternative to mechanical disks in main stream computer clusters [14]. These high capacity SSD's have implemented a sophisticated Flash Translation Layer (FTL) that achieves both high performance and reliable data durability. They exhibit many technical advantages over mechanical disks, such as outstanding random-access performance, high sequential-access throughput, low power consumption, compact form factor, and better shock resistance. Usage of SSD's increases Hadoop throughtput considerably [24]. With SSD's, the performance improvement is almost seven or eight fold.

Topology switching [16] is a method that provides control to individual applications for deciding best how to route data among their nodes. Topology switching formalizes the simultaneous use of multiple routing mechanisms in a data center, allowing applications to define multiple routing systems and deploy individualized routing tasks at small time scales. For today's data center host applications with varied networking needs, this overcomes the shortcomings of using a single-minded forwarding approach which most certainly results in letting paths go unused, sacrificing performance, or making the whole network available to all applications, sacrificing needs like isolation.

Hadoop-A [29] is an acceleration framework that optimizes Hadoop with plugin components implemented in C++ for fast data movement. The performance improvement is derived from three methods. First, a new method is used to merge the input to the reduce phase. Instead of repeated merge and store to disk, a reducer

fetches only the keys from mappers, and merges them. It fetches the value only at the end, when all the merges are completed. Since the size of the key is small compared to that of the value, the entire input for merge can be stored in memory, avoiding going to the disk. Next, a pipeline is designed to overlap the shuffle, merge and reduce phases.

Hadoop Online [25] presents a modified version of the Hadoop MapReduce framework that supports online aggregation, which allows users to see "early returns" from a job as it is being computed. The Hadoop Online Prototype (HOP) also supports continuous queries, which enable MapReduce programs to be written for applications such as event monitoring and stream processing.

3.2 Challenges in Deploying Interconnects

Hadoop does not take advantage of high-performance RDMA[5] interconnect technologies such as InfiniBand [1] that have matured in the HPC (High Performance Computing) community [29]. For example, a node in a hierarchical Hadoop cluster is typically equipped with one or more Gigabit Ethernet (GigE)[21] network interface cards and connected to the lowest tier GigE switch. With this conguration, one card can only add an upper bound of 125 MB/sec to the data movement throughput of Hadoop. Given a multi-socket and multi-core server, such capacity has to be shared and very thinly divided amongst all cores. Worse yet, advances in processor technology will soon deliver compute servers with hundreds of cores to the mass market. Furthermore, RDMA supports high bandwidth data movement with very little CPU involvement. Simply replacing the network hardware with the latest interconnect

technologies such as InfiniBand and 10 Gigabit Ethernet, and continuing to run Hadoop on TCP/IP will not enable Hadoop to leverage the strengths of RDMA. Thus, the lack of support for RDMA interconnects will become a severe threat for Hadoop to keep up with the advances of other computer technologies, particularly when more highly capable processors, storage, and interconnect devices are deployed to various computing and data centers.[2]

3.3 Cluster-based Challenges

Data Centers in the recent years host an array of applications. Applications deployed in the same Data Center share the available resources in parallel. Thus the underlying network is also shared among them. When the Hadoop job is running, other applications may also be running and carrying out their relevant data exchanges across the cluster. Thus there is a strong possibility that a large amount of data is travelling on the same wire as the Hadoop data exchange happens. This in turn will leave effectively compromised bandwidth for the Hadoop data exchange. The Hadoop Job will take more time to finish because of unavailable bandwidth during its data exchange. This reduces the efficiency of the Hadoop framework.

3.4 Network-based Challenges

Network-based challenges in Hadoop have been long known, but acceleration techniques to avoid them haven't been explored yet. In current Hadoop distribution, the Mappers select the path to reach the Reducers on a random basis. The Mappers

don't evaluate the shortest/cheapest path to reach to the Reducers. This may lead to a Mapper sending its intermediate file to a Reducer via a long/costlier path in turn the MapReduce processing taking more time to complete. As discussed above, there are a lot of file transfers happening in the underlying network. These file transfers take longer to complete because of the other traffic in the network.

In our approach, we give Mappers the ability to choose the shortest/cheapest path to reach to their selected Reducer. We also give Hadoop the ability to setup preferred Quality of Service (QoS) on the network devices in the underlying network to optimize its file transfers.

Chapter 4

Our Approach - Accelerating Data-intensive Computations through Dynamic Network Traffic Optimization

4.1 Software Defined Networking

Software Defined Networking (SDN) is revolutionizing the networking industry, offering capability for control over the network to entities that did not otherwise have any say in how their network was configured for their use. Network devices have two aspects:

1. **Control Plane** - made up of all the protocols that help decide where & how to forward data traffic, and

2. **Data Plane** - made up of the set of facilities available on the device to forward data traffic.

In traditional network devices, both control and data planes reside on the device, within what is known as the embedded network operating system (NOS). Software defined networking is a concept that separates the control plane from the data plane on the network device, so it can reside outside the embedded NOS as a separate software entity, including on any commodity server. Software defined networking centralizes the control planes of all the network devices in a given network infrastructure. Separating the control plane from the data plane on the network device provides the following benefits:

1. The control plane can be run as a separate software entity, even on a commodity server.

2. The performance and scalability of the control plane software can be increased independent of the CPU and memory capabilities of the network devices.

3. The network devices themselves become simpler, easier to manage, and potentially less expensive to build.

Centralizing the control planes of all the network devices provides the following benefits:

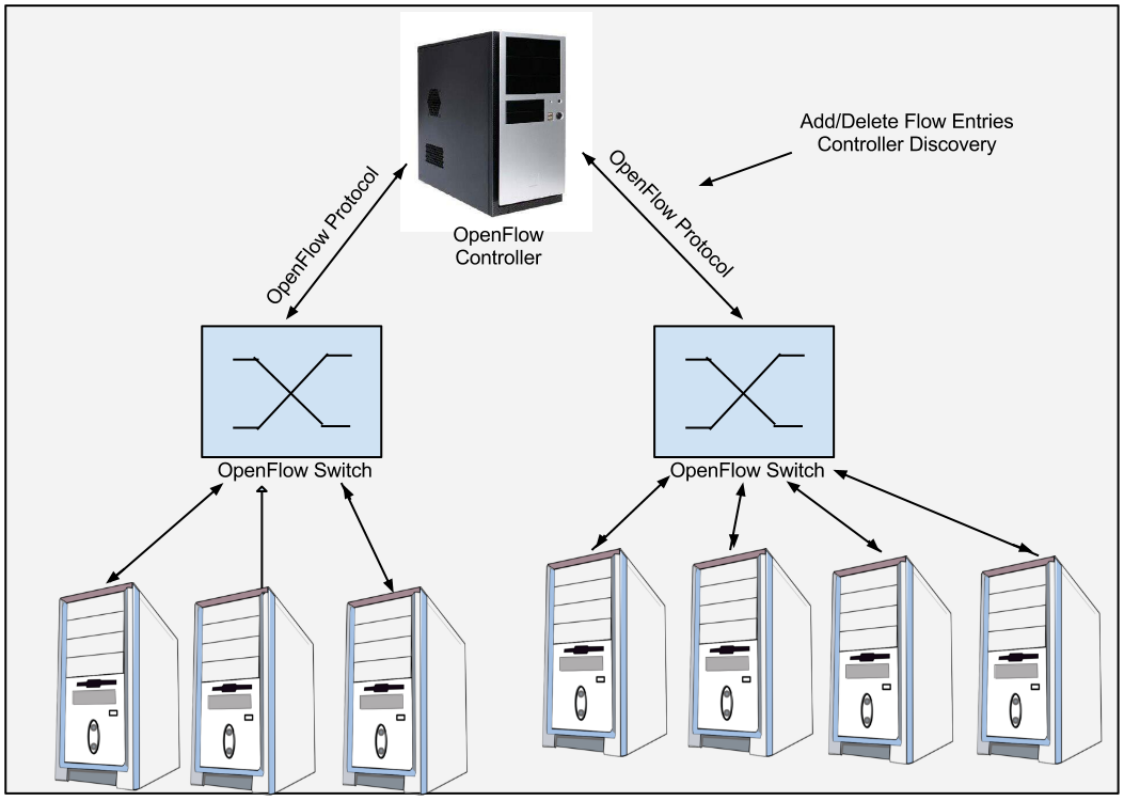

Figure 4.1: SDN Paradigm

1. It greatly simplifies the management of the individual network devices because the management complexity of a network device is typically associated with the control plane protocols that reside on it.

2. It provides opportunities for eliminating some control protocols entirely (like Spanning Tree), and allows for sophisticated traffic management without requiring any additional capability in the network devices.

4.2 OpenFlow

OpenFlow [11] is a secure communication protocol between a remote controller software and agent software that runs on a network device. It gives access to the data forwarding plane of that network device over the network. The protocol includes a well defined "forwarding instruction set" that gives the remote controller software the power to modify the behavior of each network device. A well defined, open API allows "applications" that handle control protocols and provide advanced traffic management capabilities to directly "program" the network using the remote controller. OpenFlow is an enabler of software defined networking.

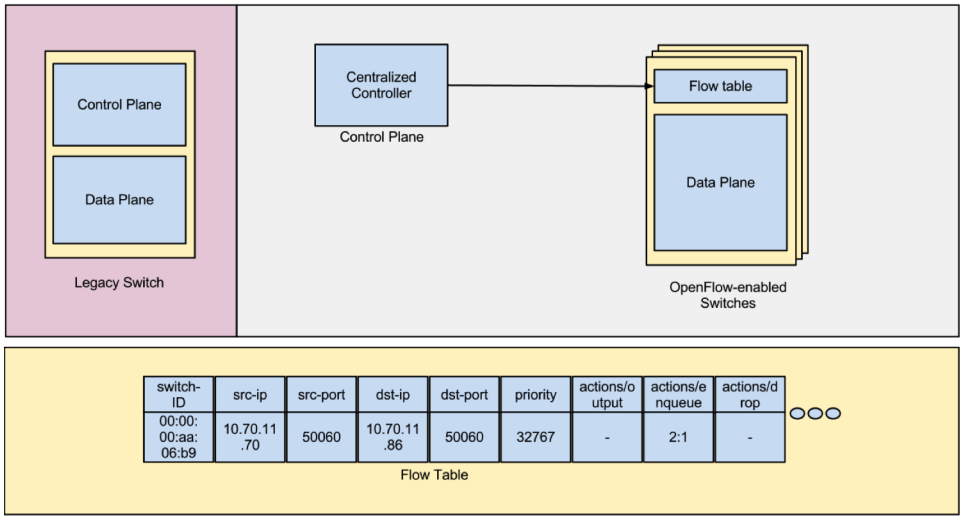

Figure 4.2: Traditional versus OpenFlow Switches

The data path portion still resides on the switch, while high-level routing decisions are moved to a separate controller. The OpenFlow Switch and Controller communicate via the OpenFlow protocol, which defines messages, such as packet-received, send-packet-out, modify-forwarding-table, and get-stats. The data path

of an OpenFlow Switch presents a clean flow table abstraction; each flow table entry contains a set of packet fields to match, and an action (such as send-out-port, modify-field, or drop). When an OpenFlow Switch receives a packet it has never seen before, for which it has no matching flow entries, it sends this packet to the controller. The controller then makes a decision on how to handle this packet. It can drop the packet, or it can add a flow entry directing the switch on how to forward similar packets in the future. Thus, OpenFlow enables one to easily deploy innovative routing and switching protocols in a network. Another feature of OpenFlow is simple Quality-of-Service (QoS). An OpenFlow switch provides limited support for QoS through a simple queuing mechanism. One or more queues can attach to a port and be used to map flows on it. Flows mapped to a specific queue will be treated according to that queues QoS configuration (e.g. minimum rate).

4.3 Dynamic Priority Assignment

When we submit a Hadoop Job, the HDFS sends out split files of the input payload to the Mappers. The Mappers process the split files and generate intermediate files to be sent to the Reducers at the end of the Map phase. The Intermediate files are sent to the Reducers, this phase is the Shuffle phase. Reducers aggregate the intermediate files and write the end result back to the HDFS. All these interactions between the Mappers and Reducers are orchestrated by the Job Tracker process which is running on the NameNode. The Job Tracker has the priori knowledge of which are the Mapper nodes and which are the Reducer nodes. The Mappers and Reducers also report all the start and end task events to the Job Tracker.

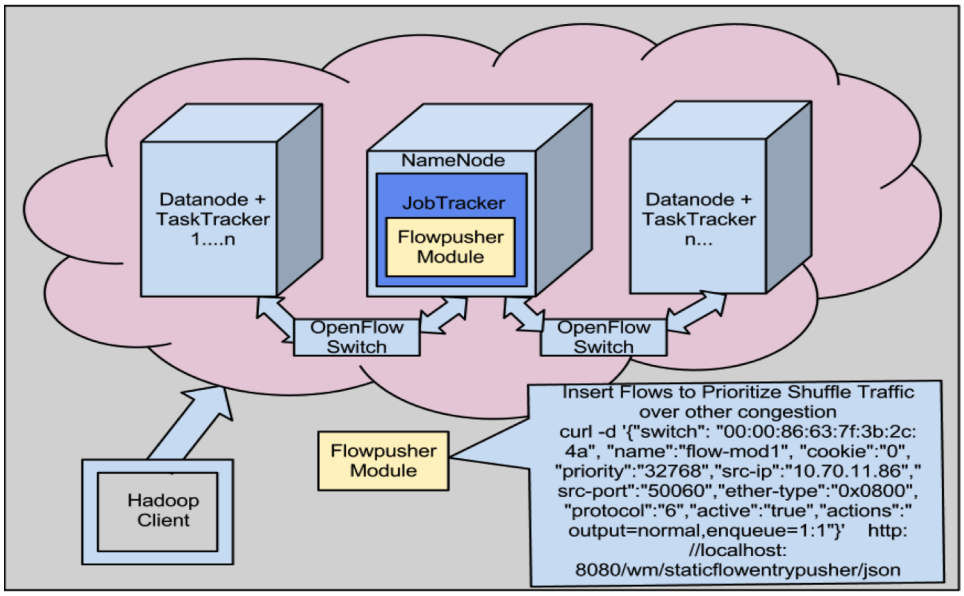

Figure 4.3: Dynamic Flow Pusher

In our implementation, we edit the Job Tracker code so that at the end of the Mapper task it inserts a flow with high priority in the underlying OpenFlow Switch. The flow inserted has the src-ip of the Mapper and the dst-ip of the Reducer. It also configures the flow for only the src-port 50060 because the Hadoop Shuffle traffic comes only out port 50060. Thus, the Shuffle traffic uses the high priority, high QoS queue for its transfer. The Job Tracker removes the flow at the end of the Shuffle phase.

There can be a phase in the Hadoop ecosystem where multiple jobs might try to push flows to the same switch at the same time. Though this situation is not taken care of in our proposed solution, we propose to solve this by implementing a lock based priority assignment. A job with higher priority will be able to request a lock on the switches in its path. Meanwhile, other jobs with lower priority will have to

wait for this lock to be released to be able to push the flows on the same switches.

4.4 Application Layer Traffic Optimization

Today, network information available to applications is mostly from the view of endhosts. There is no clear mechanism to convey information about the network infrastructure (e.g., preferences or topological properties) to applications, forcing applications to make approximations using data sources such as BGP Looking Glass or their own measurements, which can be misleading or inaccurate. On the other hand, modern network applications can be adaptive, with the potential to become more network-efficient (e.g., reduce network resource consumption) and achieve better application performance (e.g., accelerated download rate), by leveraging better network-provided information.[15]

The ALTO Service provides a simple mechanism to convey network information to applications. Its objective is to provide basic, abstract but useful network information to applications. The mechanism includes abstractions to achieve concise, flexible network information expression.[22]

At a high level, the ALTO Service allows a Service Provider (e.g., an ISP) to publish information about network locations and costs between them at configurable granularities.

Applications that use the ALTO Service can benefit in multiple ways. For example, they may no longer need to infer topology information, and some applications can reduce reliance on measuring path performance metrics themselves. They can take advantage of the ISP's knowledge to avoid bottlenecks and boost performance.

The ALTO Server provides batch information to ALTO Clients in the form of Network Map and Cost Map. The Network Map provides the full set of Network Location groupings defined by the ALTO Server and the Endpoints contained with each grouping. The Cost Map provides costs between the defined groupings.

4.5 Interface to Metadata Access Points Protocol (IF-MAP)

IF-MAP protocol [13] is published by the Trusted Computing Group (TCG) and is managed by the Trusted Network Connect (TNC) sub-group, which includes a number of vendors and end users. The TNC group has defined and implemented a suite of protocols that provide an open solution architecture that enables network operators to enforce policies. Part of the TNC architecture is IF-MAP, which defines a standard interface between any product or system and a so called Metadata Access Point, which is a database for aggregating, correlating, and distributing metadata between different systems in real time. The latest version of the IF-MAP protocol was released in August 2010 and there are a number of commercial and open-source products that include the protocol and that have been deployed in production environments.

The IF-MAP architecture is very general and can be used for use cases outside the realm of network security. Specifically, IF-MAP can be used for the protocol between the interfaces of measurement services and IF-MAP objects (identifiers, metadata and links) can be used to define a very flexible, adaptable, and immediately

implementable schema. The IF-MAP protocol is lightweight and supports three primitive operations: Publish, Subscribe, and Search. IFMAP supports a network or graph information model similar to RDF [12] but with some simplifying assumptions which makes implementation of IF-MAP clients simple. It is an XML dialect with bindings, which include SOAP/HTTPS.

4.6 Intelligent Path Determination

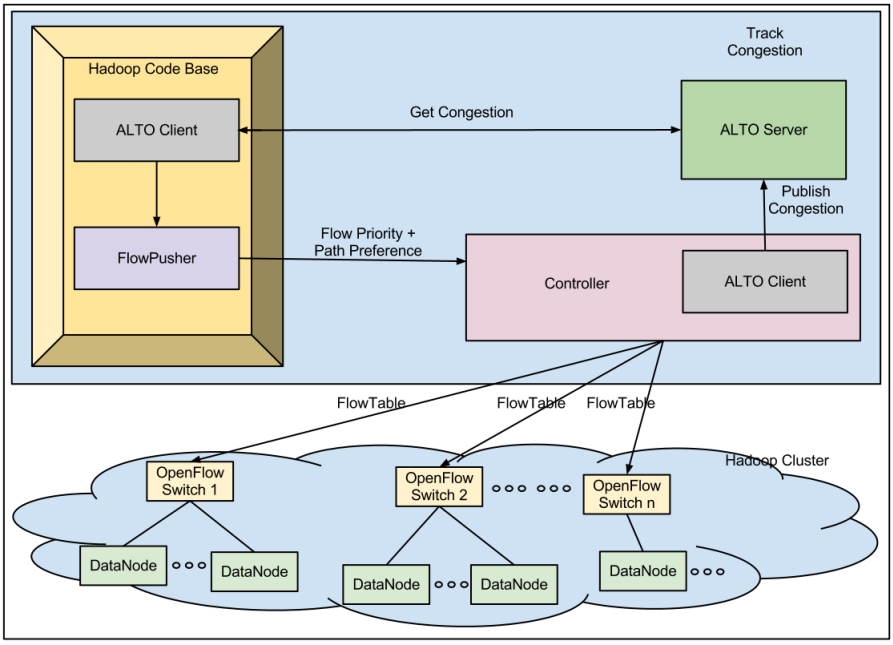

Figure 4.4: Pictorial Overview of the Overall System

In current Hadoop distribution, the Mappers select the path to reach the Reducers on a random basis. The Mappers don't evaluate the shortest/cheapest path to reach to the Reducers. This may lead to a Mapper sending its intermediate file to a Reducer via a long/costlier path in turn the MapReduce processing taking more

time to complete. As discussed above, there are a lot of file transfers happening in the underlying network. These file transfers take longer to complete because of the other traffic in the network. In our approach, we give Mappers the ability to choose the shortest/cheapest path to reach to their selected Reducer.

As illustrated in Figure 4.4, the Hadoop Cluster Manager has an embedded ALTO Client which publishes the Cluster Topology to the ALTO Server at its inception. An ALTO Client embedded inside the OpenFlow Controller, publishes the real-time stats from the OpenFlow Switches in the Cluster. The above two published information forms the ALTO Information Base which helps select the cheapest/shortest path in the Cluster.

Hadoop Job Tracker has the apriori knowledge of the roles of all the nodes in the Cluster. Job Tracker knows which nodes are Mappers and which nodes are Reducers. When the Map phase in the Hadoop Job completes, the Job Tracker knows about the Reducers where the Mappers will be sending the intermediate files. Job Tracker takes the help of the ALTO Client embedded inside the Hadoop Code Base to get the congestion information in the underlying network from the ALTO Server. Once the Job Tracker gets the congestion information, it evaluates the cheapest/shortest path to send the intermediate files to the Reducers. Thus, it decides on the path and contacts the Flow Pusher class to insert appropriate flows on the selected path.

Also, the path determination technique used in our proposed solution is very naive. Deciding the current congestion in the underlying network just depending on the packet count may not be a wiser choice. Future scope will comprise of coming up with a better decision policy where other parameters like system usage, link latency,

etc. will be considered before making the intelligent path determination.

Chapter 5

Experiment Setup

5.1 Hadoop Cluster and Network

The 15 node Hadoop cluster ran the Cloudera distribution of Hadoop [4] on 3 physical systems (Server1, Server2, and Server3) connected to a physical switch as shown in Figure 5.1. The Hadoop nodes were running on virtual machines (VMs) in a XenServer [3] virtualized environment. The Cloudera Cluster Manager (CM) ran on one of the VMs in the cluster and managed the Hadoop cluster. The Open vSwitch (OVS) [19] is the default network stack for the XenServer. Open vSwitch is a multi-layer software switch that supports OpenFlow standard 1.0 [10]. We used BigSwitchs opensource Apache-license, Java-based OpenFlow Controller called Floodlight [18] to setup flow entries in the OpenFlow switches.

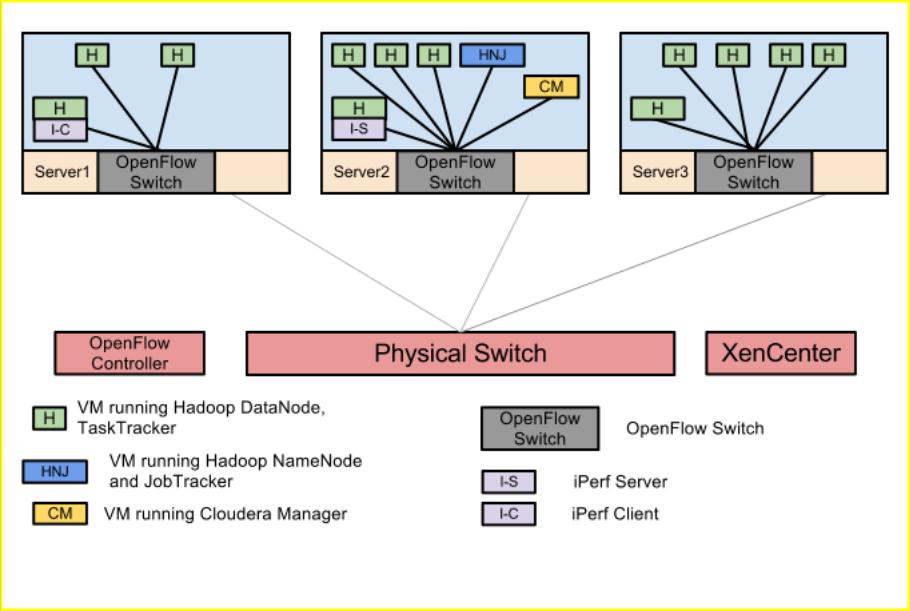

Figure 5.1: Hadoop Cluster Setup

5.2 Test Procedures

For the experiments, we used the Hadoop Sort program to run as the job under test. Hadoop comes with a MapReduce program that does a partial sort of its input. It is very useful for benchmarking the whole MapReduce system, as the full input dataset is transferred through the intermediate shuffle phase. The three steps of the Hadoop Sort program are: generate random data, perform sort, and validate the results [27].

For the first set of experiments we used the QoS capability of the OpenFlow standard so that we could explore the use of OpenFlow in a small cluster with just three systems. First, we setup the maximum rate (QoS) on all the three OpenFlow vSwitches to be 300 Mbps. Next, we created three queues, the first (q0) with 50

Mbps, second (q1) with 200 Mbps, and the third (q2) with 5 Mbps. Even though our network supported higher bandwidth, we used these rates to artificially create congestion and investigate the benefit of OpenFlow switches under such situations. We generated additional traffic in the cluster using a utility called Iperf [26]. We ran an Iperf server on one VM and a client on another VM on a different physical system. In all cases, we ran Hadoop Sort jobs for the different input payloads (size: 0.4 MB, 4 MB, 40 MB, 400 MB and 4GB).

In the first experiment (Experiment 1), we did not insert any flows in the OpenFlow switches. Thus, all the traffic traveled through the queue q0. No other significant traffic was present in the cluster. We recorded the time taken for each sort job.

In the second experiment (Experiment 2), we ran the Hadoop job in parallel with the Iperf traffic which created significant congestion in the network. All the traffic traveled through the queue q0. Again, we ran the Hadoop Sort job on the input payloads and recorded the time taken for each sort job.

In the next experiment (Experiment 3), we used OpenFlow to provide different network characteristics to some traffic categories of Hadoop traffic. In Hadoop, the Task Trackers communicate with each other using HTTP on port 50060. The major part of this interaction between Task Trackers is due to the shuffle phase where intermediate data is moved from Mappers to Reducers. We added new flow entries in all the OpenFlow switches to direct all traffic on port 50060 to queue q1, which had highest bandwidth. Iperf traffic was directed to queue q2, the lowest bandwidth queue, and the rest of the traffic used queue q0. We ran the same Hadoop Sort job

for the same input payloads and recorded the time taken for each sort job.

5.3 Sort Benchmark

Sort is often used as a baseline benchmark for HDFS. The sorting program has been pervasively accepted as an important performance indicator of MapReduce [24], because sorting is an intrinsic behavior of the MapReduce framework. In our experiments, the input data of Sort is generated using the RandomWriter. At the beginning of its execution, Sort maps the tasks to all data nodes. Each task then reads local data and performs sorting in parallel. A reduce phase follows the sorting phase to merge the locally-sorted data chunks from all data nodes and writes them to a new file.

5.4 ALTO Prototype

ALTO Prototype comprises of ALTO Server and Clients. ALTO Server has been built by leveraging the graph database and pub-sub capabilities of the IF-MAP Server. A topology publish script (ALTO Client) publishes the network topology of our experimental Hadoop to the ALTO Server. We have generated switch stats using normal distribution in MATLAB. These stats include Packet Count, Byte Count, and Flow Count on the OpenFlow Switches in the Cluster. A stats publish script (ALTO Client) publishes these stats on the ALTO Server. The above two published information forms the ALTO Information Base.

Now we run Flow Pusher script where we manually pass arguments about the

Mapper and the Reducer. This Flow Pusher script queries the ALTO Server to get the network map and cost map. By processing the network map and cost map it evaluates which will be the cheapest/shortest path to send the intermediate file to the Reducer from the Mapper. Depending on the path selected, flows are pushed to the corresponding OpenFlow vSwitches to establish preferred QoS.

Mininet [17] tool is used in this prototype to create the OpenFlow-based network. IF-MAP Server is used to host the ALTO Information Base. IF-MAP Perl Framework [13] is used to design the ALTO Clients.

5.5 Integration Considerations for ALTO-Hadoop

We used IF-MAP Server and Protocol to implement ALTO Server and clients because it provides graph database structure and pub-sub capability. Graph database makes it easier to maintain the network topology. Pub-sub gives the ALTO clients the ability to consult the ALTO Server for network related information.

So, to integrate the proposed ALTO solution with Hadoop the only requirement is to include IF-MAP modules in the Hadoop Library.

Chapter 6

Results

6.1 Without ALTO Results Discussion

Table 6.1: Hadoop Performance with OpenFlow

I/P Payload Size (MB)	Time Taken w/o Iperf (sec)	With Congestion (sec)	With Congestion & OF-enabled (sec)
0.4	16.66	23.66	22
4	19.33	25	21.33
40	22.66	28.33	26
400	55	73.66	69.66
4000	270.66	446	426.66

Table 6.1 shows the results of experiments 1, 2, and 3 described in Chapter 5. Column 2 has the results for experiment 1, Column 3 has the results for experiment 2, and Column 4 has the results for experiment 3. It shows that the Hadoop job performs better when Iperf traffic is assigned to a queue with lower maximum bandwidth, and the rest of the traffic is assigned to queue with higher maximum bandwidth. It also shows the improvement in performance seen for the case where

Hadoop shuffle traffic is assigned to the queue with higher bandwidth.

Results in the above table show the average of 3 readings taken per input payload for each of the 3 experiments. Below graphs in Figures 6.1 and 6.2, show the distribution of the readings taken in the experiments.

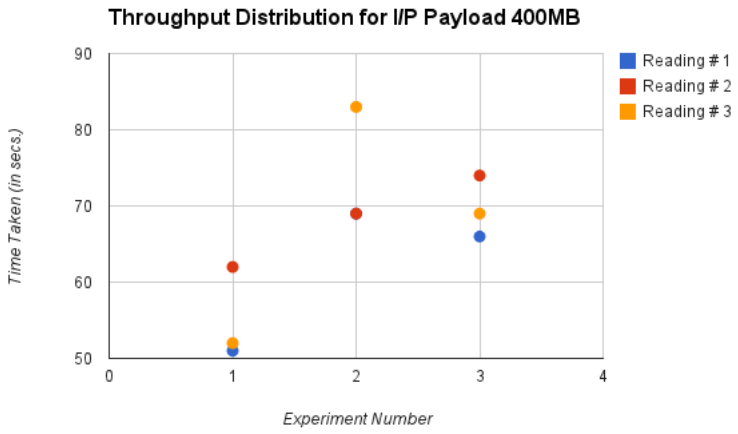

Figure 6.1: Throughput Distribution for I/P Payload 400 MB

6.2 With ALTO Anticipated Outcomes

As discussed above, if there are multiple paths to the Reducer, in current distribution of Hadoop, the Mapper randomly selects one path to direct traffic to the desired Reducer. There is a strong possibility that the chosen path might be costlier/longer than other available paths. So, when the Mapper consults the ALTO Server to evaluate and choose the best possible cheapest/shortest path, it can be expected to avoid the delay which was possible by using other paths. If we just compute shortest path, we can expect to achieve results analogous to the Shortest Path First Protocol.

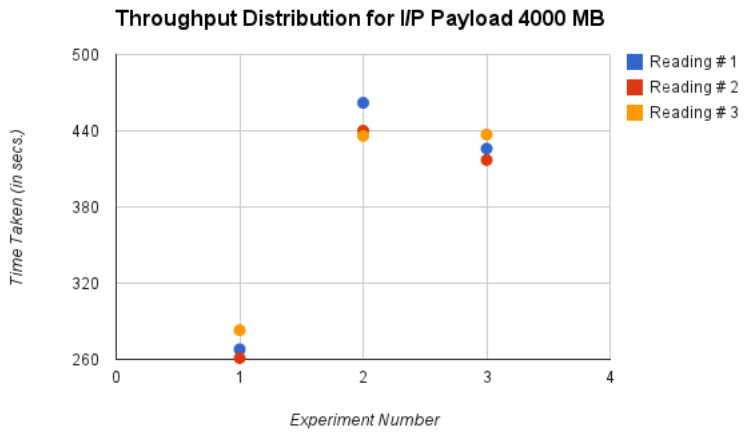

Figure 6.2: Throughput Distribution for I/P Payload 4000 MB

Chapter 7

Conclusion & Future Scope

7.1 Conclusion

We have described OpenFlow and ALTO enabled Hadoop and several experimental studies we performed to quantify the performance advantages of a version of Hadoop that uses OpenFlow and ALTO to dynamically adjust the network topology and gain increase in its throughput.

We have tried to determine that applications in the Data Centers, is the prime domain to leverage the offerings of the Software Defined Paradigm.

From our study it is evident that not all Hadoop Jobs can benefit from the acceleration schema we have proposed but only those jobs who generate a lot of shuffle traffic will benefit. Also, we have not been able to deduce under what circumstances (like bigger input payload, bigger underlying network fabric, etc.) will the percentage gain in throughput be more.

7.2 Future Scope

As we analyze the results we can see that there is a considerable increase in Hadoop throughput by just optimizing the shuffle traffic. On the same lines, we can also setup dedicated high bandwidth flows for:

1. HDFS Replication.
2. Hadoop's inter-cluster replication.
3. The Split Phase in Hadoop.

Coming up with a well-defined function to process the Network Map and Cost Map from the ALTO Information Base in order to evaluate the cheapest/shortest path will also be a good study.

Implementing rendezvous type Mapper and Reducer selection in the Hadoop distribution will also help optimize Hadoop.

Bibliography

[1] Infiniband Trade Association. Infiniband specification document. http://www.infinibandta.org, 2010.

[2] A. Penmetsa G. Bradski C. Ranger, R. Raghuraman and C. Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. *IEEE Intl Symp. on High Performance Computer Architecture (HPCA)*, pages 13–24, 2007.

[3] Citrix. Xenserver 6.0. http://blogs.citrix.com/2011/09/30/xenserver-6-0-is-here/, 2012.

[4] Cloudera. Cloudera distribution of hadoop. https://ccp.cloudera.com/display/SUPPORT/Downloads.

[5] RDMA Consortium. Architectural specications for rdma over tcp/ip. http://www.rdmaconsortium.org, 2009.

[6] J. Dean and S. Ghemawat. Mapreduce: simplied data processing on large clusters. *OSDI04: Proceedings of the 6th conference on Symposium on Opearting Systems Design and Implementation. USENIX Association*, 2004.

[7] S. Ghemawat W. C. Hsieh D. A. Wallach M. Burrows T. Chandra A. Fikes F. Chang, J. Dean and R. E. Gruber. Bigtable: A distributed storage system for structured data. *OSDI06: Seventh Symposium on Operating System Design and Implementation. USENIX Association*, 2006.

[8] Apache Software Foundation. Hadoop distributed file system (hdfs). *Apache HDFS Architecture Guide*, 2010.

[9] Apache Software Foundation. Apache hadoop nextgen mapreduce (yarn). *Apache YARN Architecture Guide*, 2011.

[10] Open Networking Foundation. Openflow switch specification. *ONF Specifications Version 1.0.0 (Wire Protocol 0x01)*, 2009.

[11] Open Networking Foundation. Software-defined networking: The new norm for networks. *ONF White Paper*, 2012.

[12] RDF Working Group. Rdf: Resource description framework. http://www.w3.org/RDF/, 2004.

[13] Trusted Computing Group. Trusted network connect (tnc) if-map binding for soap, specification version 2.1. http://www.trustedcomputinggroup.org/resources/, 2012.

[14] Solid State Storage Initiative. Solid state storage 101. *SNIA White Paper*, 2009.

[15] E. Burger J. Seedorf. Application-layer traffic optimization (alto) problem statement. *Internet Engineering Task Force, Internet-Draft draft-ietf-alto-problem-statement-01*, 2009-10.

[16] K. Yocum K. C. Webb, A. C. Snoeren. Topology switching for data center networks. *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, 2011.

[17] B. Lantz. Mininet: rapid prototyping for software defined networks. http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet, 2011.

[18] BigSwitch Networks. Floodlight open sdn controller. http://floodlight.openflowhub.org/, 2011.

[19] Nicira. Open vswitch: An open virtual switch. http://openvswitch.org/, 2010.

[20] M. Olson. Hadoop: Scalable, flexible data storage and analysis. *Cloudera White Paper*, 2010.

[21] H. V. Shah P. Balaji and D. K. Panda. Sockets vs rdma interface over 10-gigabit networks: An in-depth analysis of the memory trafc bottleneck. *Workshop on Remote Direct Memory Access (RDMA): Applications, Implementations, and Technologies (RAIT), in conjunction with IEEE Cluster*, 2004.

[22] Y. Yang R. Alimi, R. Penno. Alto protocol. *Internet Engineering Task Force, Internet-Draft draft-ietf-alto-protocol-13*, 2012-13.

[23] S. Radia S. Konstantin, K. Hairong and R. Chansler. The hadoop distributed le system. *IEEE Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[24] J. Huang X. Ouyang D.K. Panda S. Sur, H. Wang. Can high-performance interconnects benet hadoop distributed file system? *Workshop on Micro Architectural Support for Virtualization, Data Center Computing, and Clouds, in Conjunction with MICRO*, 2010.

[25] P. Alvaro J. Hellerstein K. Elmeleegy R. Sears T. Condie, N. Conway. Mapreduce online. *NSDI*, pages 313–328, 2010.

[26] The Iperf team. Iperf. http://iperf.sourceforge.net/, 2010.

[27] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media / Yahoo Press, Sebastopol, CA, USA, 2010.

[28] Wikipedia. Applications of hadoop. http://en.wikipedia.org.

[29] W. Yu Goldenberg D. Sehgal D. Y. Wang, X. Que. Hadoop acceleration through network levitated merge. *High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–10, 2011.