A Project Report

ON

Effective Message Passing on Hadoop

BY

Surabhi R
USN: 1PI10IS109
&
Shreya Chippagiri
USN: 1PI10IS134

Guide
Mr. Dinesh Singh
Assistant Professor,
Department of CSE,
PESIT
Bangalore

August 2013 – Dec 2013

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

PES INSTITUTE OF TECHNOLOGY
100 FEET RING ROAD, BANASHANKARI III STATE
BANGALORE-560085

PES Institute of Technology,
100 Feet Ring Road, BSK 3rd Stage, Bangalore-560085

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CERTIFICATE (Centre, Bold, Underlined –16size)

Certified that the project work entitled **Effective Message Passing on Hadoop** is a bonafied work carried out by **Surabhi R (USN: 1PI10IS109) AND Shreya Chippagiri (USN: 1PI10IS134)** in partial fulfillment for the award of degree of Bachelor of Engineering in Information Science of PES Institute of Technology, Bangalore during the year 2013.

Mr. Dinesh Singh **Dr. S.S. Shylaja**

Name of the Students with Reg.No:

1. **Surabhi R (USN: 1PI10IS109)**
2. **Shreya Chippagiri (USN: 1PI10IS134)**

TABLE OF CONTENTS

1. INTRODUCTION

A MapReduce program is composed of a **Map()** procedure that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a **Reduce()**procedure that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

The model is inspired by the map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as in their original forms. Furthermore, the key contributions of the MapReduce framework are not the actual map and reduce functions, but the scalability and fault-tolerance achieved for a variety of applications by optimizing the execution engine once.

MapReduce is as a 5-step parallel and distributed computation:

1. **Prepare the Map() input**– the "MapReduce system" designates Map processors, assigns the K1 input key value each processor would work on, and provides that processor with all the input data associated with that key value.
2. **Run the user-provided Map() code**– Map() is run exactly once for each K1 key value, generating output organized by key values K2.
3. **"Shuffle" the Map output to the Reduce processors**– the MapReduce system designates Reduce processors, assigns the K2 key value each processor would work on, and provides that processor with all the Map-generated data associated with that key value.
4. **Run the user-provided Reduce() code** – Reduce() is run exactly once for each K2 key value produced by the Map step.
5. **Produce the final output** – the MapReduce system collects all the Reduce output, and sorts it by K2 to produce the final outcome.

MapReduce libraries have been written in many programming languages, with different levels of optimization. A popular open-source implementation is Apache Hadoop.

Apache Hadoop is an open-source software framework for storage and large scale processing of data-sets on clusters of commodity hardware. Hadoop is an Apache top-level project being built and used by a global community of contributors and users.It is licensed under the Apache License 2.0.

All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are common and thus should be automatically handled in software by the framework. Apache Hadoop's MapReduce and HDFS components originally derived respectively from Google's MapReduce and Google File System(GFS) papers.

Beyond HDFS, YARN and MapReduce, the entire Apache Hadoop “platform” is now commonly considered to consist of a number of related projects as well – Apache Pig, Apache Hive, Apache HBase, and others.

For the end-users, though MapReduce Java code is common, any programming language can be used with "Hadoop Streaming" to implement the "map" and "reduce" parts of the user's program. Apache Pig, Apache Hive among other related projects expose higher level user interfaces like Pig latin and a SQL variant respectively. The Hadoop framework itself is mostly written in the Java programming language, with some native code in C and command line utilities written as shell-scripts.

2. **PROBLEM STATEMENT**

Effective Message Passing on Hadoop.

Message passing amongst the slave nodes and between the master and the slave nodes can lead to tremendous network traffic, bottlenecks and latencies if not optimised.

With this project we attempt to,

1. Enumerate the Network bottlenecks and latencies in existing systems.

2. Measure the performance by testing against standardized benchmarks.

3. Arrive at a possible theoretical solution to optimize the Network latencies using Link State Algorithm.

4. Optimize the network routes using a Software Defined Network(SDN) approach to obtain scalability for a large data set by harnessing MapReduce and HDFS.

3. LITERATURE SURVEY

1. **Overcoming Networking Bottlenecks of Hadoop MapReduce Jobs by, William Clay Moody December 4, 2012.**

   Recent advances in parallel and distributed computing has created a new area of research for computer scientists, creatively named Big Data. Big data applications run in warehouse sized data centers filled with rows of racks of commodity computer hardware that run large processing jobs on huge amounts of data. A popular big data application is Apache Hadoop, which is an open source implementation of Google MapReduce. Studies of Hadoop clusters show a network bottleneck impacting the performance of MapReduce jobs. Two recent publications have introduced creative approaches to addressing this bottleneck to improve cluster performance. In this paper, I will survey both of those approaches and introduce new research studying a specific characteristics of MapReduce jobs needed to implement both optimization approaches. My results show that skew in the partitioning of reducer input data does exist in well know MapReduce jobs and can potentially be predicted and modeled to implement both optimization tactics.

2. **Center-of-Gravity Reduce Task Scheduling to Lower MapReduce Network Traffic, Authors:Mohammad Hammoud, M. Suhail Rehman, Majd F. Sakr**

   MapReduce is by far one of the most successful realizations of large-scale data-intensive cloud computing platforms. MapReduce automatically parallelizes computation by running multiple map and/or reduce tasks over distributed data across multiple machines. Hadoop is an open source implementation of MapReduce. When Hadoop schedules reduce tasks, it neither exploits data locality nor addresses partitioning skew present in some MapReduce applications. This might lead to increased cluster network traffic. In this paper we investigate the problems of data locality and partitioning skew in Hadoop. We propose Center-of-Gravity Reduce Scheduler (CoGRS), a locality-aware skew-aware reduce task scheduler for saving MapReduce network traffic. In an attempt to exploit data locality, CoGRS schedules each reduce task at its center-of-gravity node, which is computed after considering partitioning skew as well. We implemented CoGRS in Hadoop-0.20.2 and tested it on a private cloud as well as on Amazon EC2. As compared to native Hadoop, our results show that CoGRS minimizes off-rack network traffic by averages of 9.6% and 38.6% on our private cloud and on an Amazon EC2 cluster, respectively. This reflects on job execution times and provides an improvement of up to 23.8%.

3. **Programming Your Network at Run-time for Big Data Application by, Guohui Wang?, T. S. Eugene Ngy, Anees Shaikh, IBM T.J. Watson Research Center, Rice University.**

   Bin-packing placement: We use rack-based bin-packing placement for reduce tasks to aggregate them onto a minimum number of racks. Since the SDN controller configures topology and routing at the rack level, rack based bin-packing placement can create opportunities to aggregate traffic on these racks and reduce the number of ToR switches that need to be configured.

   Batch processing: The network configuration of map and reduce tasks should be handled in batches, where a group of tasks submitted in a period T will be processed together. Within a batch of tasks, the job tracker selects those with greatest estimated volume and requests the SDN controller to set up the network for them. Tasks in earlier batches have higher priority such

that tasks in later batches can be allocated optical circuits only if there are left over circuits from earlier batches. There are two benefits of doing batch processing. First, it helps aggregate traffic from multiple jobs to create long duration traffic that is suitable for circuit switched paths. Second, it ensures that an earlier task does not become starved by later tasks. The batch processing can be implemented as a simple extension to Hadoop job scheduling.

4. **Hadoop Performance Models, Technical Report,Computer Science Department, Duke University.**

   Hadoop MapReduce is now a popular choice for performing large-scale data analytics. This technical report describes a detailed set of mathematical performance models for describing the execution of a MapReduce job on Hadoop. The models describe data flow and cost information at the fine granularity of phases within the map and reduce tasks of a job execution. The models can be used to estimate the performance of MapReduce jobs as well as to find the optimal configuration settings to use when running the jobs.

5. **Hadoop Cluster Applications - Arista Whitepaper**

   Data analytics has become a key element of the business decision process over the last decade. Classic reporting on a dataset stored in a database was sufficient until recently, but yesterday's data gathering and mining techniques are no longer a match for the amount of unstructured data and the time demands required to make it useful. The common limitations for such analysis are compute and storage resources required to obtain the results in a timely manner. While advanced servers and supercomputers can process the data quickly, these solutions are too expensive for applications like online retailers trying to analyze website visits or small research labs analyzing weather patterns. Hadoop is an open-source framework for running data-intensive applications in a processing cluster built from commodity server hardware. Some customers use Hadoop clustering to analyze customer search patterns for targeted advertising. Other applications include filtering and indexing of web listings, or facial recognition algorithms to search for specific images in a large image database, to name just a few. The possibilities are almost endless provided there is sufficient storage, processing, and networking resources.

6. **Benchmarking and Stress Testing an Hadoop Cluster With TeraSort, TestDFSIO & Co. – Michael Noll**

   In this article I introduce some of the benchmarking and testing tools that are included in the Apache Hadoop distribution. Namely, we look at the benchmarks TestDFSIO, TeraSort, NNBench and MRBench. These are popular choices to benchmark and stress test an Hadoop cluster. Hence knowing how to run these tools will help you to shake out your cluster in terms of architecture, hardware and software, to measure its performance and also to share and compare your results with other people

7. **A Benchmarking Case Study of Virtualized Hadoop Performance**

   The performance of three Hadoop applications is reported for several virtual configurations on VMware vSphere 5 and compared to native configurations. A well-balanced seven-node AMAX ClusterMax system was used to show that the average performance difference between native and the simplest virtualized configurations is only 4%. Further, the flexibility enabled by virtualization to create multiple Hadoop nodes per host can be used to achieve performance significantly better than native.

4. PROJECT REQUIREMENTS DEFINITION

SYSTEM REQUIREMENTS:

Operating System: Ubuntu 12.04 LTS
RAM: 4 GB
HARD DISK: 100 GB or more

Hadoop Version: 1.2.1

5. SOFTWARE REQUIREMENT DEFINITION

Hardware Interface:

Describes the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, communication protocols to be used.

Hardware Requirements:

Processor : Intel i5 Pocessor

RAM : 4GB

Hard Disk : 500 GB

Software Interface:

Describes the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.

Software Requirements:

Front End : Hadoop, Apache Web Server,Ganglia,Nagios

Operating System : Ubuntu 12.04 LTS

6. SYSTEM DESIGN

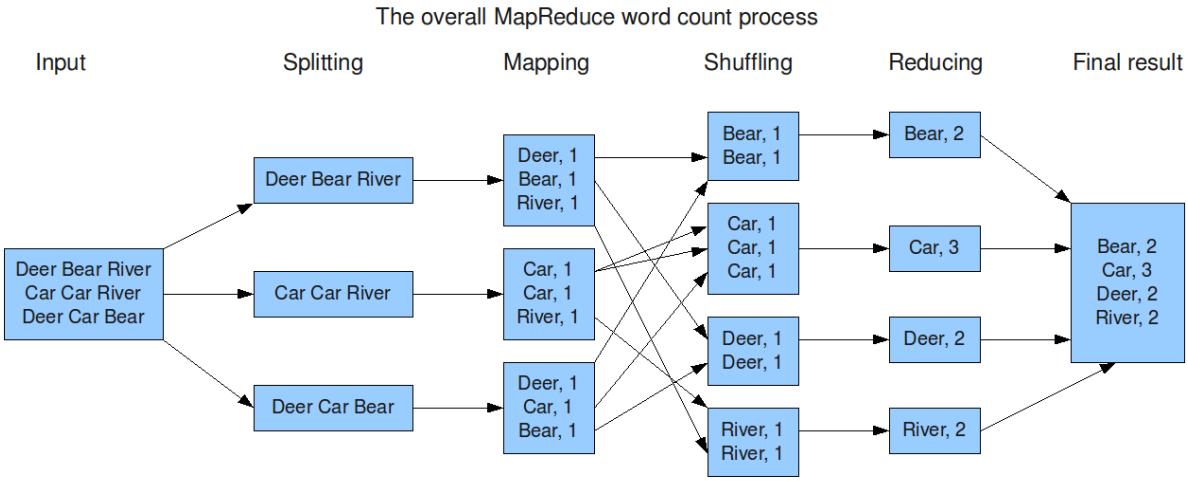


Fig 1.1 MAP REDUCE

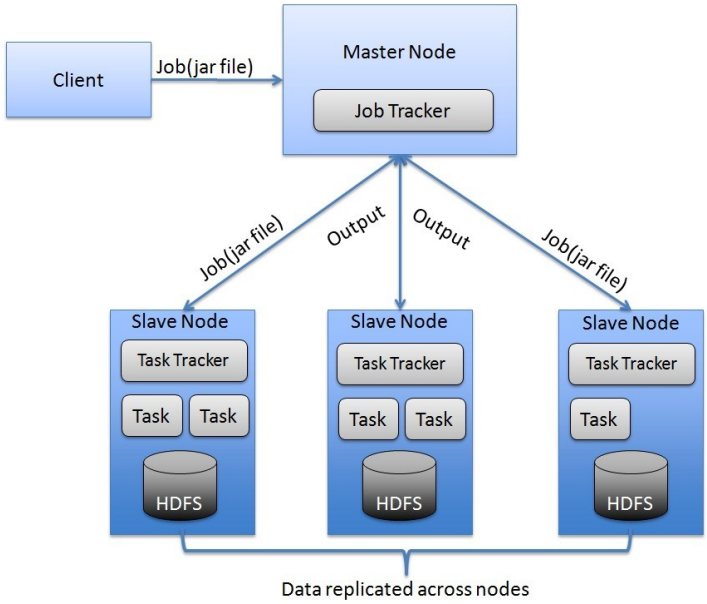


Figure 1.2 Hadoop Master-Slave Architecture

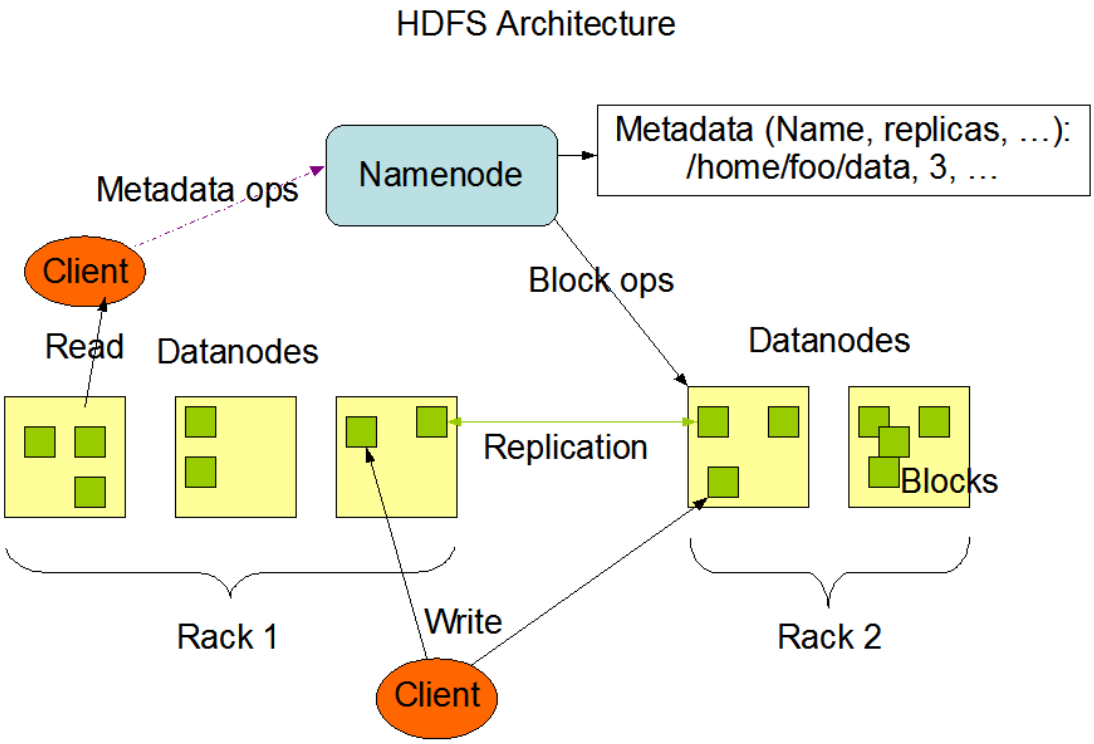


Figure 1.3 HDFS ARCHITECTURE

7. PSEUDO CODE/PROCEDURE

TESTING BENCHMARKS:

DFSIO is part of the hadoop distribution and can be found in "hadoop-mapreduce-client-jobclient-*-tests.jar" for MR2. There are two types of DFSIO tools – TestDFSIO and TeraSort. TestDFSIO is Distributed I/O Benchmark tool as per the help description below. There are several options you can pass into the tool.

Benchmarking IO Performance

Step 1: $ hadoop jar $HADOOP_HOME/hadoop-test.jar TestDFSIO -write -nrFiles 10 -fileSize 1000

Step 2: $ hadoop jar $HADOOP_HOME/hadoop-test.jar TestDFSIO -read -nrFiles 10 -fileSize 1000
It will also write a log file to the folder from where you issue the command.

Step 3: $ hadoop jar $HADOOP_HOME/hadoop-test.jar TestDFSIO –clean
These commands will data write to /benchmarks folder. You can delete this by issuing

Benchmarking Map-Reduce using Sort/Tera Sort Examples

Step 1: Invoke RandomWriter (in examples JAR) to write random data output to a directory called random-data:
$hadoop jar $HADOOP_INSTALL/hadoop-examples.jar randomwriter random-data
Dtest.randomwrite.bytes_per_map=5000000 -Dtest.randomwrite.total_bytes=50000000

Step 2: Next run the sort example to sort the data in “random-data” and store it into “sorted-data”. The time to sort this data will give you an idea of how efficient your cluster is.
$hadoop jar $HADOOP_INSTALL/hadoop-examples.jar sort random-data sorted-data

Step 3: A final verification can be done using the testmapredsort (SortValidator) program.
$hadoop jar $HADOOP_INSTALL/hadoop-test.jar testmapredsort -sortInput random-data -sortOutput sorted-data

Throughput mb/sec for a TestDFSIO job using N map tasks is defined as follows. The index 1<= i <= N denotes the individual map tasks:

$$Throughput(N) = \sum_{i=0}^{N} filesize_i \sum_{i=0}^{N} time_i$$

Average IO rate mb/sec is defined as:

$$Average\ IO\ rate(N) = \sum_{i=0}^{N} rate_i N = \sum_{i=0}^{N} filesize_i time_i N$$

LINK STATE ALGORITHM:

We are implementing Link State Advertisements, which basically means to communicate to my neighbors and every other node in my graph, through the link/s I'm immediately connected to by sending LSA packets.(Hence the name Link State) The link/s I'm immediately connected to also tells me which of the nodes are my immediate neighbors. To communicate I'm connected to my neighbor ports (calculated using neighbor:cost info given in the command line argument). We have 'n' number of terminals, bound to a port each.

We first start building the graph with our own information(entered through command line) for the edge costs from myself to my neighbors and back. This we call the initial matrix.

To enable us to send and receive packets simultaneously on the socket we use the select module has to be used. The socket is bound to MY port number, calculated as a base+ID of the node. (12340 + node ID in our case)

The user can send and receive packets at the same time:

At the **RECEIVER's end**
I receive the message from the socket and process(break, analyse, and update my matrix) this message only if it hasn't been received earlier.
The message is analysed, and from this message we extract the actual sender of the message, each of its neighbors, their respective cost and update the matrix with the cost values respectively.

At the **SENDER's end**
Firstly, for myself, a message is to be created, that may be circulated amongst all nodes in graphs. This message as mentioned is to be created with the values 'my node ID;number of my neighbors;time this msg is created;edge costs to my neighbors(1:2,3:4,...)'
Once this message is created, I broadcast this message to my immediate neighbors on the ports we calculated.
This message needs to be sent ONLY once to all my immediate neighbors.

Forwarding messages received
When I received messages, I would have stored them if they were unique. Each of these messages needs to be forwarded to all my neighbors, except the actual sender of the message. This enables nodes not connected to each other to update their matrices on the basis of messages being forwarded to them from their respective neighbors.

Broadcasting my packets needs to be done periodically using a time interval, whose value we take in using command line arguments.
We require identical cost matrices, at each of the 'n' terminals and ONLY then our program should stop. That is each node, should know the cost matrix for the entire graph.

8. RESULTS

After reading a variety of Research Papers, the possible bottlenecks and their solution include:

1. Network Bottleneck
    a. Why it is caused:
        1. Until all the map jobs are completed,reduce jobs cannot start.
        2. At times of failure-jobs are reassigned.
        3. A particularly slow map job can affect the subsequent jobs.
        4. Mutli-tenancy.
        5. Data-outputs arrive in bursts,which can lead to packet loss if buffers are not large enough.

    b. Possible Solution:
        1. Implement buffers of larger size.
        2. Effective job scheduling.
        3. Priority queues.
        4. Improve the performance of map-reduce tasks.

2. Network Latency
    a. Why its caused:
        i. Huge network traffic.
        ii. Large distances between nodes.(more hops).
        iii. Less bandwidth.
        iv. If jobs are busy, can affect the time complexity.

    b. Possible Solution:
        i. Larger network bandwidth considering cost/performance measures.
        ii. Optimize routes.

3. Distance between nodes
    a. Possible Solution:
        i. Rack-awareness.
        ii. SDN based network routing
        iii. Implement routing algorithms.
            1. Link State Algorithm

Benchmark results:

Write Test

13/13/13 16:16:50 INFO fs.TestDFSIO: —– TestDFSIO —– : write
13/13/13 16:16:50 INFO fs.TestDFSIO: Date & time: Fri Dec 06 16:16:50 PDT 2013
13/13/13 16:16:50 INFO fs.TestDFSIO: Number of files: 100
13/13/13 16:16:50 INFO fs.TestDFSIO: Total MBytes processed: 100000.0
13/13/13 16:16:50 INFO fs.TestDFSIO: Throughput mb/sec: 6.0895988251458775
13/13/13 16:16:50 INFO fs.TestDFSIO: Average IO rate mb/sec: 6.641181468963623
13/13/13 16:16:50 INFO fs.TestDFSIO: IO rate std deviation: 1.9043254369666331
13/13/13 16:16:50 INFO fs.TestDFSIO: Test exec time sec: 390.825
13/13/13 16:16:50 INFO fs.TestDFSIO:

Read Test

13/13/13 16:23:01 INFO fs.TestDFSIO: —– TestDFSIO —– : read
13/13/13 16:23:02 INFO fs.TestDFSIO: Date & time: Fri Dec 13 16:23:01 PDT 2013
13/13/13 16:23:02 INFO fs.TestDFSIO: Number of files: 100
13/13/13 16:23:02 INFO fs.TestDFSIO: Total MBytes processed: 100000.0
13/13/13 16:23:02 INFO fs.TestDFSIO: Throughput mb/sec: 18.524110055442662
13/13/13 16:23:02 INFO fs.TestDFSIO: Average IO rate mb/sec: 20.380735397338867
13/13/13 16:23:02 INFO fs.TestDFSIO: IO rate std deviation: 6.731484273400149
13/13/13 16:23:02 INFO fs.TestDFSIO: Test exec time sec: 171.87

COMPARISON OF LINKSTATE ALGORITHM AND DISTANCE VECTOR ALGORITHM

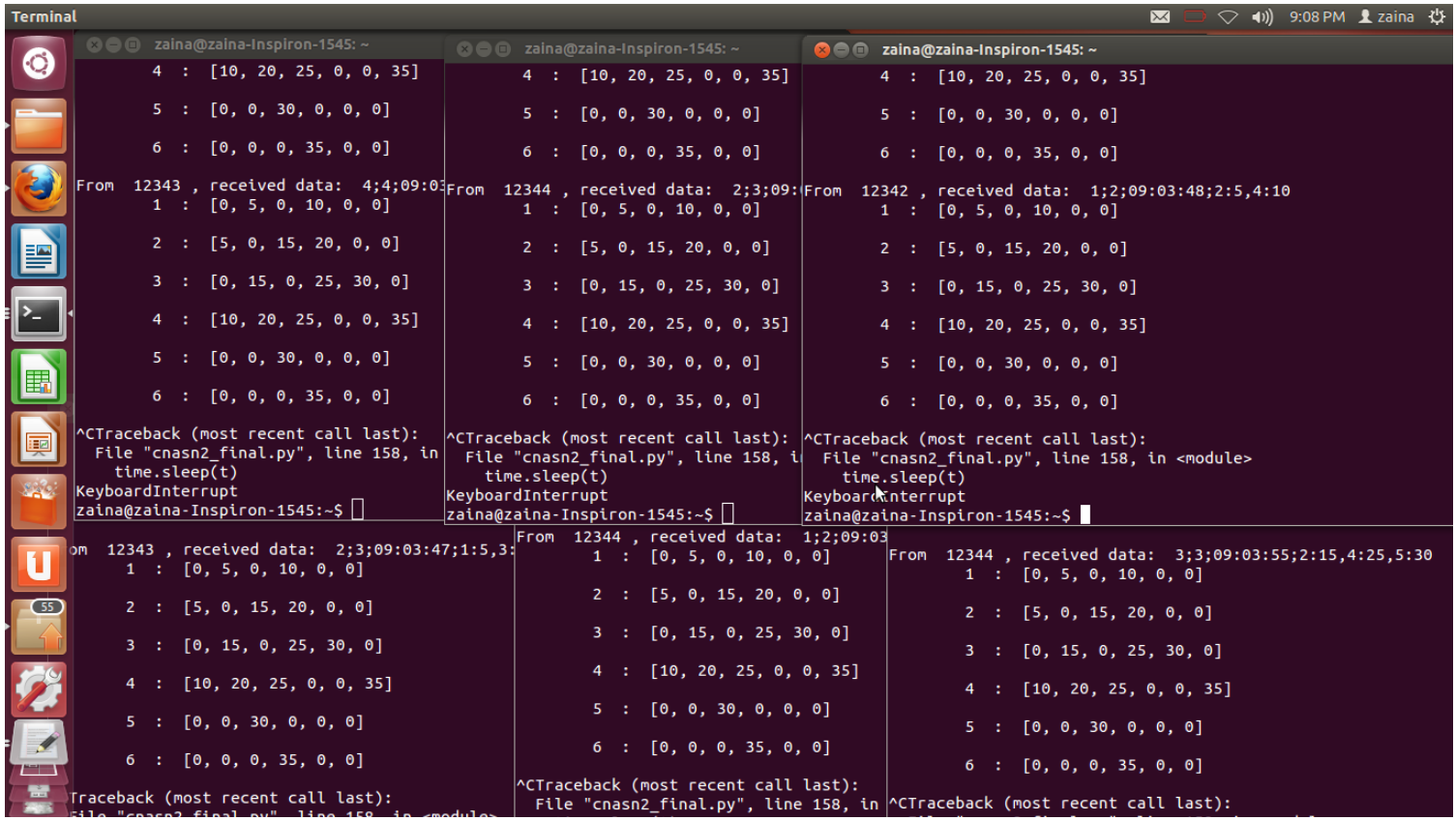
Link State Algorithm: Every node constructs a map of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes. Each node then independently calculates the next best logical path from it to every possible destination in the network. The collection of best paths will then form the node's routing table.

Distance-vector protocols are based on calculating the next hop address and the cost to reach a certain node. The least cost route between any two nodes is the route with minimum distance. Each node maintains a vector table of minimum distance to every node. The cost of reaching a destination is calculated using various route metrics like RIP uses the hop count of the destination. IGRP takes into account other information such as node delay and available bandwidth.

Message complexity: With link state, every node has to keep the information about the cost of each link within the network. And every times, if any of the cost is changed, all the nodes. With distance vector algorithm, message is exchanged between two hosts which are directly connected to each other. And if the change of cost in the link which is belong to the least cost path for one of the nodes, the DV algorithm will update the new value. But if the change doesn’t belong to the least cost part between 2 hosts, there will no updating.

Speed of convergence: the implementation of LS is an **O(|N|2)** which need **O(|N||E|)** message. But with the DV algorithm, it can converge slowly and have routing loops while the algorithm is converging. In addition, the DV algorithm also suffers from the count to infinity problem.

Robustness: For LS, when a router is down, it can broadcast a wrong cost for the closest one. And also, a node can corrupt or drop packet it get as part of a LS broadcast. However, an LS node is computing for its own forwarding table and other node do the calculation for themselves. So it makes the calculation separated in some way within LS which provide robustness. For DV, the wrong least cost path can be passed to more than one or all of the node so the wrong calculation will be process in the entire network. This problem of DV is much worse than LS algorithm.


Terminal
9:08 PM zaina
zaina@zaina-Inspiron-1545: ~
4 : [10, 20, 25, 0, 0, 35]
5 : [0, 0, 30, 0, 0, 0]
6 : [0, 0, 0, 35, 0, 0]
From 12343 , received data: 4;4;09:03
1 : [0, 5, 0, 10, 0, 0]
2 : [5, 0, 15, 20, 0, 0]
3 : [0, 15, 0, 25, 30, 0]
4 : [10, 20, 25, 0, 0, 35]
5 : [0, 0, 30, 0, 0, 0]
6 : [0, 0, 0, 35, 0, 0]
^CTraceback (most recent call last):
File "cnasn2_final.py", line 158, in
time.sleep(t)
KeyboardInterrupt
zaina@zaina-Inspiron-1545:~$
zaina@zaina-Inspiron-1545: ~
4 : [10, 20, 25, 0, 0, 35]
5 : [0, 0, 30, 0, 0, 0]
6 : [0, 0, 0, 35, 0, 0]
From 12344 , received data: 2;3;09:
1 : [0, 5, 0, 10, 0, 0]
2 : [5, 0, 15, 20, 0, 0]
3 : [0, 15, 0, 25, 30, 0]
4 : [10, 20, 25, 0, 0, 35]
5 : [0, 0, 30, 0, 0, 0]
6 : [0, 0, 0, 35, 0, 0]
^CTraceback (most recent call last):
File "cnasn2_final.py", line 158, i
time.sleep(t)
KeyboardInterrupt
zaina@zaina-Inspiron-1545:~$
zaina@zaina-Inspiron-1545: ~
4 : [10, 20, 25, 0, 0, 35]
5 : [0, 0, 30, 0, 0, 0]
6 : [0, 0, 0, 35, 0, 0]
From 12342 , received data: 1;2;09:03:48;2:5,4:10
1 : [0, 5, 0, 10, 0, 0]
2 : [5, 0, 15, 20, 0, 0]
3 : [0, 15, 0, 25, 30, 0]
4 : [10, 20, 25, 0, 0, 35]
5 : [0, 0, 30, 0, 0, 0]
6 : [0, 0, 0, 35, 0, 0]
^CTraceback (most recent call last):
File "cnasn2_final.py", line 158, in <module>
time.sleep(t)
KeyboardInterrupt
zaina@zaina-Inspiron-1545:~$
om 12343 , received data: 2;3;09:03:47;1:5,3:
1 : [0, 5, 0, 10, 0, 0]
2 : [5, 0, 15, 20, 0, 0]
3 : [0, 15, 0, 25, 30, 0]
4 : [10, 20, 25, 0, 0, 35]
5 : [0, 0, 30, 0, 0, 0]
6 : [0, 0, 0, 35, 0, 0]
Traceback (most recent call last):
From 12344 , received data: 1;2;09:03
1 : [0, 5, 0, 10, 0, 0]
2 : [5, 0, 15, 20, 0, 0]
3 : [0, 15, 0, 25, 30, 0]
4 : [10, 20, 25, 0, 0, 35]
5 : [0, 0, 30, 0, 0, 0]
6 : [0, 0, 0, 35, 0, 0]
^CTraceback (most recent call last):
File "cnasn2_final.py", line 158, in
From 12344 , received data: 3;3;09:03:55;2:15,4:25,5:30
1 : [0, 5, 0, 10, 0, 0]
2 : [5, 0, 15, 20, 0, 0]
3 : [0, 15, 0, 25, 30, 0]
4 : [10, 20, 25, 0, 0, 35]
5 : [0, 0, 30, 0, 0, 0]
6 : [0, 0, 0, 35, 0, 0]
^CTraceback (most recent call last):

9. CONCLUSIONS

Thus,several approaches can be taken to overcome the network bottlenecks of Hadoop to ensure effective message passing on it. Buffering,rack-awareness and using links with larger bandwidths are some of the already employed mechanisms. Hadoop on open-stack(cloud) can be employed to harness SDN-based network optimization.

10. FUTURE ENHANCEMENTS

Our discussion has been focused on configuring physical topology and routing for big data applications. Several other issues remain to be explored in future work to realize the full capability of the integrated network control.

Fairness, priority and fault tolerance: On the fault tolerance aspect, the SDN controllers, such as Neutron have built in mechanisms to handle network device and link failures. Failed devices and links will be updated in the topology by the SDN controller. Most big data applications have also been designed to handle failures. For example, Hadoop job tracker monitors the progression of all the jobs and failed tasks will be rescheduled onto different servers. Therefore, in the integrated system, the failure handling mechanisms can remain untouched with application managers and the SDN controller handling failures at different levels. Since network configuration is performed over batches of application tasks, task-level fairness and priority scheduling within an application can be done by the application job scheduler. However, the data center network infrastructure is normally shared by multiple applications. Application managers could request network configuration concurrently for their tasks. The fairness and priority among different application requests must be handled by the SDN controller. The scheduling policy of SDN controller for different applications is an open question to be explored.

Traffic engineering for big data applications: Using OpenFlow protocol, SDN controller can perform flow-level traffic engineering for big data applications. There are multiple routes on the rack level going through different optical paths and electrical paths. Accurate traffic demand and structural pattern from applications can allow SDN controller to split or re-route management and data flows on different routes. Implementing flow level traffic engineering requires installing a rule for each selected flow on ToR switches, which imposes additional overhead to the network configuration. In future work, we will explore flow level traffic engineering mechanisms for big data applications and the efficient implementation of them using SDN controller.

11. BIBLIOGRAPHY

Websites:

1. Openstack-www.openstack.org
2. Devstack-www.devstack.org
3. Hadoop-hadoop.apache.org
4. ask.openstack.org

Research Papers:

1. Overcoming Networking Bottlenecks of Hadoop MapReduce Jobs William Clay Moody - wcm@clemson.edu December 4, 2012 http://www.chinacloud.cn/upload/2012-12/12122409238948.pdf
2. Performance Issues of Heterogeneous Hadoop Clusters in Cloud Computing http://arxiv.org/abs/1207.0894
3. Meta-data driven data ingestion using MapReduce framework https://www.google.co.in/patents/EP2653968A2?cl=en&dq=hadoop+mapreduce&hl=en&sa=X&ei=o2WGUpygFIToiAfHi4CIAw&ved=0CFMQ6AEwAw
4. Center-of-Gravity Reduce Task Scheduling to Lower MapReduce Network Traffic, Authors:Mohammad Hammoud, M. Suhail Rehman, Majd F. Sakr
5. Programming Your Network at Run-time for Big Data Application by, Guohui Wang?, T. S. Eugene Ngy, Anees Shaikh, IBM T.J. Watson Research Center, Rice University.
6. Hadoop Performance Models, Technical Report,Computer Science Department, Duke University.
7. Hadoop Cluster Applications - Arista Whitepaper
8. Benchmarking and Stress Testing an Hadoop Cluster With TeraSort, TestDFSIO & Co. – Michael Noll
9. A Benchmarking Case Study of Virtualized Hadoop Performance