Hadoop Acceleration in an OpenFlow-based cluster

Sandhya Narayan, Stu Bailey
InfoBlox Inc.
Santa Clara, U.S.A
snarayan@infobox.com

Anand Daga
Dept. of Computer Science, Univ. of Houston
Houston, U.S.A
adaga2@cs.uh.edu

Abstract*— This paper presents details of our preliminary study of how Hadoop can control its network resources using OpenFlow in order to improve performance. Hadoop's distributed compute framework called MapReduce, exploits the distributed storage architecture of Hadoop's distributed file system HDFS to deliver scalable, reliable parallel processing services for arbitrary algorithms. The shuffle phase of Hadoop's MapReduce computation involves movement of intermediate data from Mappers to Reducers. Reducers are often delayed due to inadequate bandwidth between them and the Mappers, and thereby lower the performance of the cluster. OpenFlow is a popular example of software-defined network (SDN) technology. Our study explores the use of OpenFlow to provide better link bandwidth for shuffle traffic, and thereby decrease the time that Reducers have to wait to gather data from Mappers. Our experiments show decrease in execution time for a Hadoop job, when the shuffle traffic can use more of the available bandwidth on a link. Our approach illustrates how high performance computing applications can improve performance by controlling their underlying network resources.

The work presented in this paper is a starting point for some experiments being done as part of SC12 SCinet Research Sandbox which will quantify the performance advantages of a version of Hadoop that uses OpenFlow to dynamically adjust the network topology of local and wide area Hadoop clusters.

***Index Terms*—Hadoop, BigData, OpenFlow, Software Defined Networks (SDN).**

I. Introduction

Hadoop has emerged as a prominent 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. Its distributed storage system called HDFS is a single, consolidated storage platform with a new type of repository where structured data and complex data may be combined easily. The analysis system called MapReduce framework, exploits the distributed storage architecture of HDFS to deliver scalable, reliable parallel processing services for arbitrary algorithms [1].

It is evident that HDFS's performance characteristics are very much dependent on the performance of disk I/O. Likewise, MapReduce Framework's performance depends on the compute power of the cluster. However, the data movement in Hadoop clusters is a major factor is the overall performance of the Hadoop cluster. Even when jobs are scheduled closer to where its data lie in the cluster, network traffic is still an issue, because of inadequate bandwidth between nodes when they need it.

One approach to address the network issues in Hadoop is to provide an application developer or user with 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. Ability to control bandwidth allocation as needed by an application is expected to provide improvement in Hadoop performance just like specialized interconnects, such as InfiniBand can.

Software Defined Network (SDN) is revolutionizing the networking industry, offering a multitude of opportunities for control over the network to entities that did not otherwise have any say in how their network was configured for their use. These entities could be anyone: network operators, service providers, application developers, and even end users.

The leading SDN technology is based on the OpenFlow protocol, a standard that has been designed for SDN and already being deployed in a variety of networks and networking products, both hardware- and software-based [2]. In the OpenFlow network architecture, the control and data planes are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications. As a result, enterprises and carriers can gain unprecedented programmability, automation, and network control, enabling them to build highly scalable, flexible networks that readily adapt to changing business needs [2].

II. Overview of Hadoop MapReduce

MapReduce 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 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 job's 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.

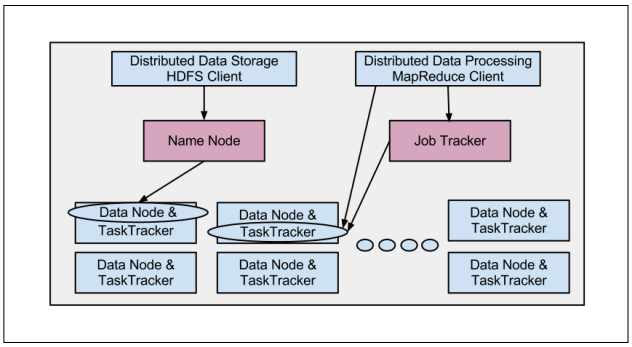

Fig. 1. Hadoop MapReduce Framework.

MapReduce's 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. The input data, that is stored in HDFS prior to beginning of the job, is 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.

Hadoop network and compute architecture considerations have been presented in [11], which explores the effect of network on Hadoop performance. The work identifies the bursty nature of Hadoop traffic, the overall network speed and Hadoop's design for reliability as some of the factors affecting Hadoop performance. Many industry and research efforts are addressing these issues. [12] explores the use of Infiniband interconnect in Hadoop clusters. [13] presents a new pipeline design that overlaps the shuffle, merge and reduce phases, along with an alternative protocol to enable data movement via RDMA (Remote Direct Memory Access). Instead, we focused on utilizing the network control capabilities provided by OpenFlow to provide resources to straggling Reducers, who can be speeded up by prioritizing their use of the network.

Network traffic in Hadoop can be separated into several categories:

- HDFS read and write by client
- HDFS replication
- Interaction between TaskTrackers, which includes Hadoop shuffle traffic
- Hadoop split traffic between HDFS and TaskTrackers
- Hadoop results stored in HDFS
- Interactions between NameNode and DataNode
- Interactions between JobTracker and TaskTracker

In addition to these traffic categories, the cluster may be multi-use cluster and may run other applications concurrently. This background traffic can also affect completion times of Hadoop-jobs [11].

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 are known a-priori and can be used to the advantage of setting up a more performant 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.

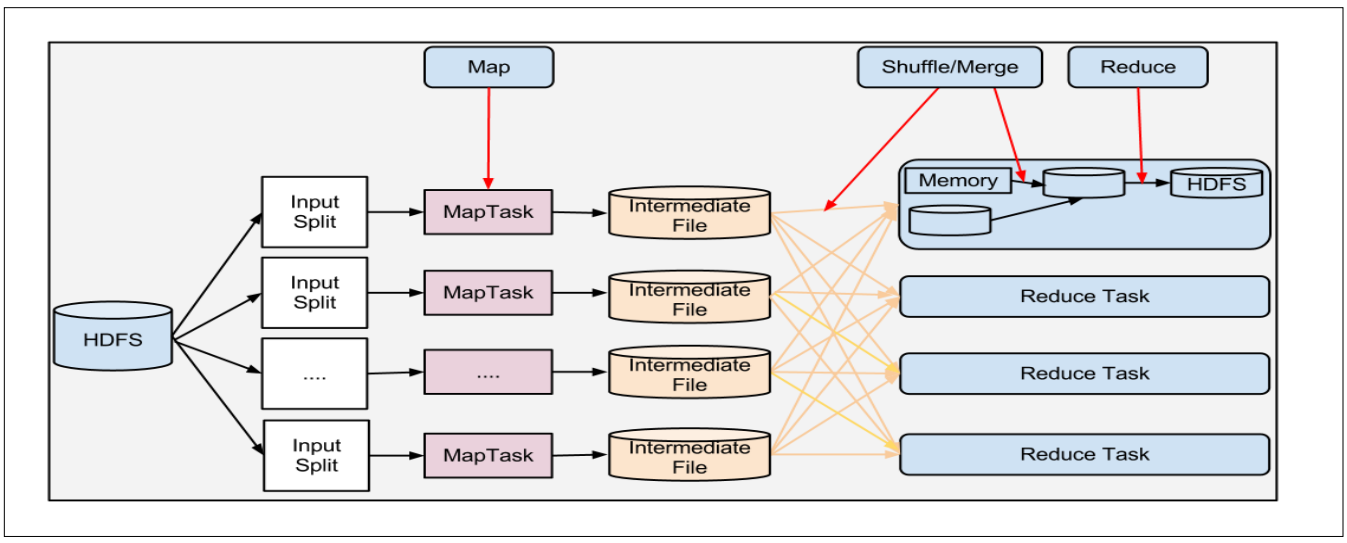

Fig. 2. Data Processing in Hadoop.

III. Overview of OpenFlow

In a classical router or switch, the data path and the control path occur on the same device. An OpenFlow Switch separates these two functions. 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, modify-forwarding-table, and get-stats, that are exchanged between the OpenFlow switch and the controller.

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

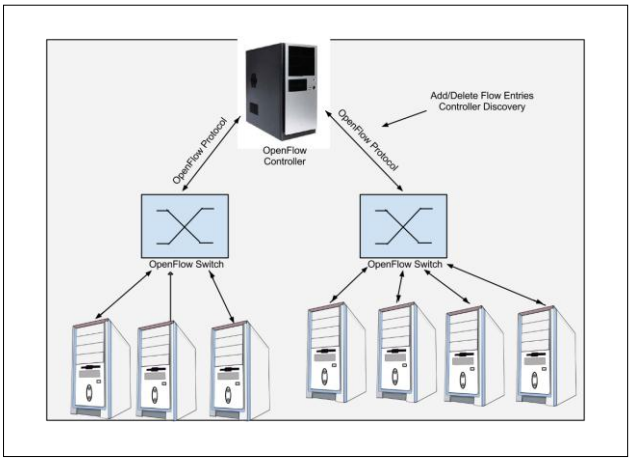

Fig. 3. OpenFlow Paradigm.

forward similar packets in the future. [4].

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 queue's QoS configuration (e.g. minimum rate).

IV. OpenFlow and Hadoop

Using the two features of OpenFlow discussed above, we can provide different network characteristics to various categories of traffic in a Hadoop cluster. Some simple examples are: a) provide Hadoop traffic with higher priority over other traffic in the cluster and b) provide Hadoop shuffle traffic with higher priority. To illustrate this we setup a simple Hadoop cluster with OpenFlow switches (Fig. 4).

The 10 node Hadoop cluster ran the Cloudera distribution of Hadoop on 3 physical systems (Server1, Server2 and Server3) connected to a physical switch (Figure 4). The Hadoop nodes were run on virtual machines (VMs) in a XenServer virtualized environment [6]. The Cloudera Manager (CM) ran on one of the VMs in the cluster and managed the Hadoop cluster.

The Open VSwitch (OVS) [7] is the default network stack for the XenServer. Open Vswitch is a multi-layer software switch that supports OpenFlow standard 1.0. We used BigSwitch's opensource Apache-license, Java-based OpenFlow Controller called Floodlight [8] to setup flow entries in the OpenFlow switches. We also ran our experiments with another OpenFlow switch, LINC, which is an open-source software switch developed in Erlang [11] that provides a full implementation of OpenFlow 1.2 standard.

V. Experiments

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 [3].

For the first set of experiments we used the QoS capability of the OpenFlow standard so that we could explore the use of

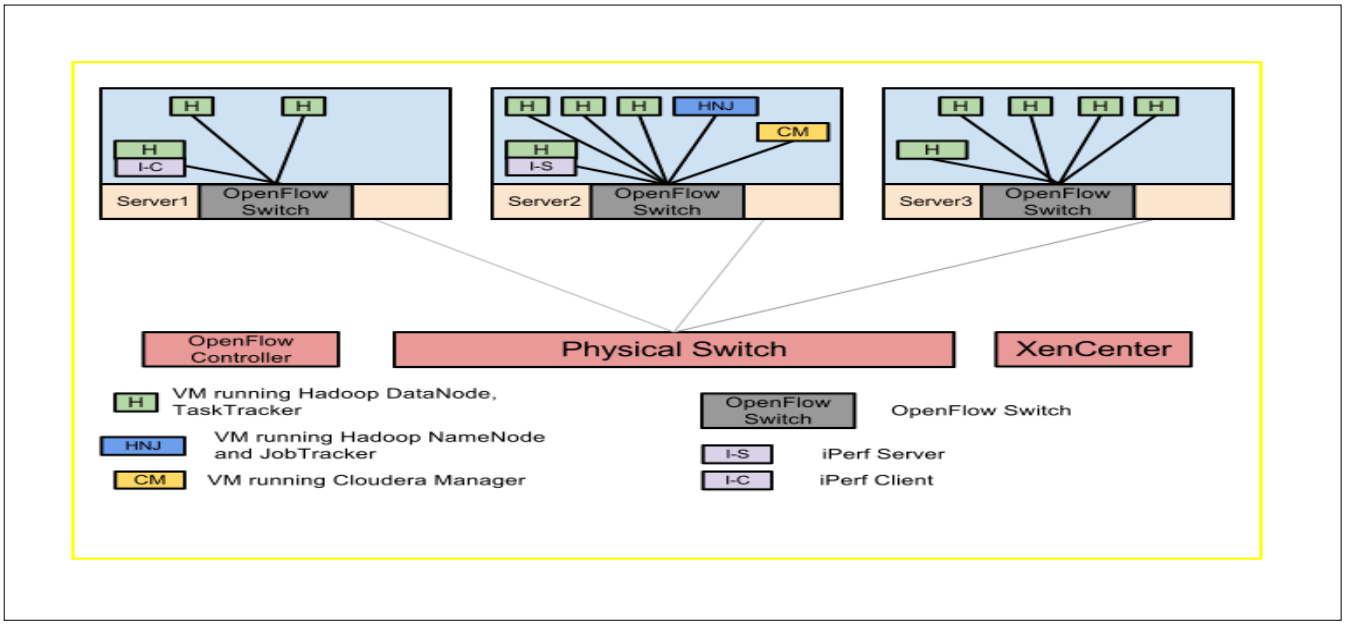

Fig. 4. Experimental Setup Architecture

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 [9]. There are several mechanisms to inject cross traffic or congestion in to the network. Some popular mechanisms to achieve this are Iperf, Scapy, Nemesis and Tcprelay. We chose Iperf as it is simple to use and sufficient for our experimental purposes to create cross traffic in the underlying network. 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 (baseline) 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 added one flow entry to each of the OpenFlow switches, which put the Iperf request traffic which uses port 5001 on queue q2 (low maximum rate). Again, we ran the Hadoop Sort job on the input payloads and recorded the time taken for each sort job.

In the third 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 higher maximum rate. Iperf traffic continued on queue q2 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.

In the next experiment (Experiment 4), we added one flow entry to each of the OpenFlow switches, which put all the traffic including Hadoop and Iperf traffic on queue q3 (highest maximum rate). Again, we ran the Hadoop Sort job on the input payloads and recorded the time taken for each sort job.

VI. Results

Figure 5 shows the results of experiments 1, 2, 3 and 4. It shows that the Hadoop job performs better when IPerf traffic is assigned to a queue with lower maximum rate and the rest of the traffic is assigned to queue with higher maximum rate. Experiment 3 shows the improvement in performance seen for the case where Hadoop shuffle traffic is assigned to the queue with a higher rate. Experiments 3 and 4 showed that Hadoop job performs better when shuffle traffic is given higher priority. In addition, we also performed the same experiments on a cluster with LINC OpenFlow switches and saw similar results.

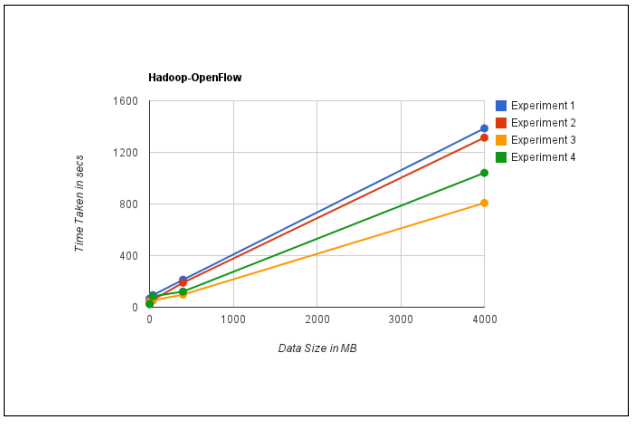

Fig. 5. Hadoop Performance with OpenFlow

We also observed that in general Hadoop jobs are long running and the reconfiguration time (in ms) to setup the switches with different flow entries is very small compared to the overall job completion times.

Our next set of experiments is in progress, where we are working to setup different topologies for different categories of traffic and to do this in a dynamic manner. We believe that this work shows promise for achieving one of the goals of software-define networking, which is to provide control over the network to applications that use the network.

References

[1] Stuart Bailey, Sandhya Narayan, Anand Daga, Robert Grossman, Matthew Greenway, OpenFlow Enabled Hadoop Over Local and Wide Area Clusters. Demo at the "SCinet Research SandBox", SuperComputing 2012.

[2] HADOOP: Scalable, Flexible Data Storage and Analysis. http://www.cloudera.com/wpcontent/uploads/2010/05/Olson_IQT_Quarterly_Spring_2010.pdf

[3] ONF, Software-Defined Networking: The New Norm for Networks. https://www.opennetworking.org/images/stories/downloads/white-papers/wp-sdn-newnorm.pdf

[4] Tom White, Hadoop: The Definitive Guide, 2nd edn., O'Reilly, Sebastopol, CA, 2011.

[5] Introduction to OpenFlow, https://www.opennetworking.org/standards/intro-to-openflow

[6] XenServer, http://blogs.citrix.com/2011/09/30/xenserver-6-0-is-here/

[7] Open Vswitch, http://openvswitch.org/

[8] FloodLight, http://floodlight.openflowhub.org/

[9] LINC Switch, http://flowforwarding.org/

[10] Iperf, http://code.google.com/p/iperf/

[11] Jacob Rapp. Hadoop Network & Compute Architecture Considerations. Hadoop World, 2011.

[12] S. Sur, H. Wang, J. Huang, X. Ouyang, and D. K. Panda. Can High-Performance Interconnects Benefit Hadoop Distributed File System? In Workshop on Micro Architectural Support for Virtualization, Data Center Computing, and Clouds (MASVDC). Dec 2010.

[13] Yandong Wang, Xinyu Que, Weikuan Yu, Dror Goldenberg, and Dhiraj Sehgal. Hadoop acceleration through network levitated merge. In Proceedings of the 2010 International Conference on Supercomputing.