Traffic Optimization for Hadoop based on SDN

Siyuan Liu, Xinyi Wang, Jiaxu Zhu
Computer Science and Engineering Department
University of California, San Diego

***Abstract*—In this paper, we implement a system based on Software-Define Network (SDN) to optimize network traffic for Hadoop shuffle stage. Derived from $FlowComb$ and $Pythia$, the idea is to build an SDN control module to adjust flow scheduling and achieve a more balanced flow allocation. The SDN controller collects the information from running mappers and reducers using an Auxiliary Process, and then updates a $Load\ Graph$ which reflects the workload in the network. The flow scheduling strategy is to avoid a routing path with heavy workload. To test our system, we simulate a set of Hadoop jobs and observe the task completion time. We compared our system with $Pythia$ and baseline (fixed switching network without SDN), finding that our system outperformed baseline by 20% to 30% and got nearly the same performance as $Pythia$.**

Keywords*—*Hadoop, SDN, Virtualization

I. Introduction

Hadoop is an open-source software framework for storing and running computing applications on a cluster of computers or hosts. Nowadays, many tasks of companies and research associations are accomplished through taking use of Hadoop. For the sake of efficiency and economy, people request a Hadoop cluster system with better performance. The performance of Hadoop relies on many factors. The basic factor is the algorithm of Hadoop, like how the mapper and reducer work and how the Hadoop controller distributes tasks to mappers/reducers. Since Hadoop is deployed on a cluster of computers, the network for the cluster obviously affects the transmission of data and control information. In intuition, even though the mappers and reducers can work very fast, the network with high latency and drop rate which cause a long transmission time will lead to bad performance in a Hadoop system.

Our concern is focused on the network of Hadoop system. According to some basic experiment, the time of maps and reduces only occupies about 30% of the completion time for one Hadoop job. It means network transmission takes lots of time in a Hadoop job. The fact gives us an opportunity to improve the performance of Hadoop system.

Network for Hadoop system is much different from the network we use for daily life. The features of Hadoop systems and applications make the network maintain some properties. We talk about the features of Hadoop and the network in the Section V. If we can catch and take use of the properties and information, there will be a more efficient network for Hadoop jobs whose completion time mainly comes from maps and reduces. This is why we consider about using SDN for the Hadoop cluster. Software-Defined Networking (SDN) is an approach to allow a network controller to manage the network scheduling in order to take fully use of the resources and improve the performance of the softwares running on the network. SDN decouples the decisions where the traffic is sent (control plane) from how to forward the traffic to the destination (data plane). As to Hadoop applications, SDN aims at a better traffic scheduling by considering how the mappers and reducers work and how the tasks are distributed.

Our idea of traffic optimization for Hadoop based on SDN derives from $FlowComb$ and $Pythia$. They build an SDN control module for the Hadoop cluster system which collect the information from running mappers and reducers and then adjust flow scheduling for the routers and switches in the network. Our system architechture is designed according to the same idea, but we made some modification and adjustment for some algorithm details and implementation of the system. It is mostly concerned about SDN for Hadoop that what kinds of information should be collected and how the network and flows are scheduled. On the whole, we collect information like the start/end time of hadoop map/recude tasks on each node and the activities of HDFS. Then the SDN controller analyses the information and updates a $Load\ Graph$ held in the controller and deploy new flow scheduling rules to the switches and routers regularly. The $Load\ Graph$ reflects the load level for the facilities in the network. Our goal is to schedule flows avoiding the nodes with large load.

We implement our system with several useful softwares which help us to simulate a Hadoop cluster system and SDN control module. The softwares are depicted in the Section VI. We take $completion\ time$ for Hadoop jobs as the only critical metric for the performance of Hadoop on SDN. But we also analyse some variables, like the rate of time of maps and reduces in the total completion time. We made series of experiment on our system. We test our system with different data size and Hadoop jobs in order to observe how is the performance on different Hadoop applications. Also we compared our system with $Pythia$ and baseline (normal network without SDN). Actually we got almost same performance with $Pythia$ and better than the baseline.

The rest of this paper is structured as follows. We review and put our work in the context of related work in Section II. Architect of Hadoop on SDN for our system is depicted in Section III. We describe more details of the design of Hadoop cluster system (Section IV) and traffic optimization by SDN (Section V). At last, we show the implementation of our entire system and the experiments in Section VI. And conclusion and further discussion are in Section VII.

II. Related Work

As computing ability grows, network traffic in cloud computing is becoming a more important reason for holding back performances. Thus, many researchers start to focus on optimizing against network bottlenecks caused by heavy workloads. Considering the data-movement parts in big data applications, Wang et al.[1] optimize the runtime network to accelerate MapReduce jobs. The researchers use an appropriate abstraction framework to interface applications with the infrastructure, such as Coflow [2] or MROrchestrator [3].

There are a lot of ways to improve network communication, for example communication scheduling is a popular one. Systems such as Orchestra [4] and Seawall propose to improve the performance of the shuffle phase by scheduling flows using a weighted fair sharing scheme rather than the default fair sharing mechanism of TCP.

Among the all related work on this topic, we pay extra attention to two systems in our project, called FlowComb [5] and Pythia[6]. FlowComb employs shuffle-phase communication intention to apply intelligent, ahead-of-flow-occurrence network optimization towards MapReduce performance improvement.

Pythia is another system which shares a lot in common with FlowComb. They both focus on the communication-heavy phase of MapReduce and try to optimize the traffic. Pythia employs real-time communication intent prediction for Hadoop and uses this predictive knowledge to optimize the data center network at runtime, aiming at Hadoop MapReduce acceleration. Besides, it employs fine-grained control of the underlying datacenter network. The results are really exciting, with significant acceleration of the Hadoop workloads under test (up to 46%).

These two prototypes provide deep insights into optimization over traffic in Hadoop. They are of significant importance guiding us to do our project. We implemented the idea of using SDN controllers to schedule flows more reasonably in a Hadoop job, according to FlowComb and Pythia.

III. Architecture of Hadoop on SDN

At a high level of abstraction, our system included three components shown in Figure 1:

1) Hadoop cluster with SDN. This part is mainly deployment of Hadoop master/slave nodes. For the sake of SDN, we replace normal routers and switches with open source switches.

2)Instrument Process (we call it Auxiliary Process) runs on each node who collects information of the Hadoop node and sends it to the SDN control system.

3)SDN control system. This module is the core of our system architecture. It consists of two parts: routing computation and flow allocation. After collecting the information from auxiliary processes, the control module executes routing computation to check if the topology of the network changes, then the flow allocation part will update flow allocation rules according to the $LoadGraph$ which is gotten from the analysis of information collected.

The logical relations of the three components pf the system architecture are described as follows. The Hadoop cluster can work without SDN control. We deploy Hadoop nodes on a set of virtual machines. And we create a $Fat\ Tree$ network topology (ex. Figure 2)for the Hadoop nodes. The Fat Tree topology eliminates the influence from bottleneck links and take use of more available resources in the network. By choosing the fatness of links, the network can efficiently use any bandwidth made available by packaging and communications technology. After the building of Hadoop cluster system, we can deploy a auxiliary process on each Hadoop slave node which monitors the activities here, like the tasks distributed and finished and the HDFS activities. The auxiliary processes send information collected on the node to the SDN control system and keep monitoring. When the Hadoop cluster system with open source switches and auxiliary processes ready, the SDN control system can work to schedule more efficient network transmission. First, the event collector collect all the information sent from the auxiliary processes on the Hadoop nodes. The event collector parses the information as different inputs, such as $< inf\_type, node\_n, task\_header, timestamp >$ and $< inf\_type, hdfsaccess, timestamp >$, for the routing computation and flow allocation. The routing algorithm for our system is k-shortest paths. Unless the network topology changes, the routing computation happens once at the start of the whole system. With k-shortest paths available, the goal of flow allocation is to distribute packets on the paths with low load which avoid high latency and drop rate. Because finding optimal flow distribution on a graph a NP-complete problem, our system computes a set of next hops for one packet with the help of $Load\ Graph$. More details about network scheduling will be depickted in Section V.

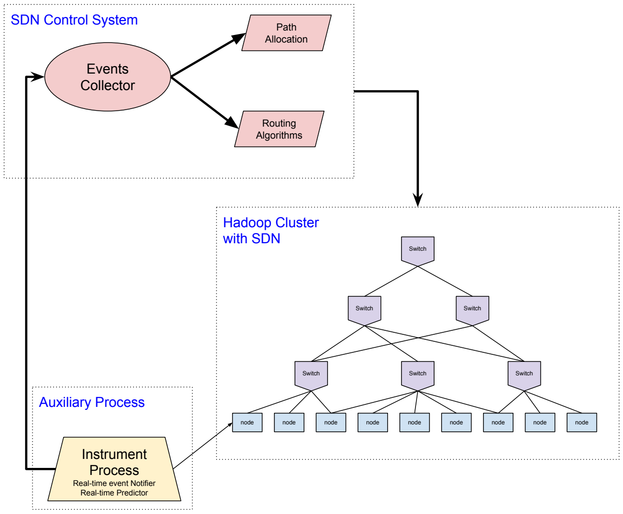

Fig. 1. Architecture of our system design

The goal of using SDN is to get fast and efficient transimission. The most important thing is to monitor the status of the current networking and generate feedback to control the following network flows. Our system obeys the intuition which tries to collect information from the running Hadoop nodes and compute new flow allocation rules. Our experiment shows that it is a positive cycle between collection and adjustment.

IV. Virtual Hadoop Cluster

A. Introduction to Hadoop

Hadoop MapReduce is a open source of Googles MapReduce [7] distributed data-processing framework. It is gaining increasing traction and deployment base in various domains

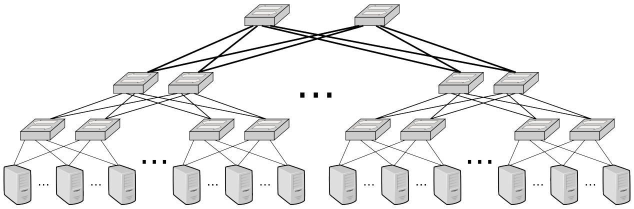

Fig. 2. Fat Tree Topology

requiring batch-processing large-scale analytics, primarily due to linear system scalability, support for both structured and unstructured data sources, a fairly straightforward programming model and transparent to the user runtime parallelism. A remarkably versatile set of applications can be implemented by using the two primitive functions of the MR model, namely map and reduce. The map function ingests input data records (values) and maps them to an application-specific key-space. In turn, the output (intermediate map output in Hadoop terminology) of the map function is sorted and fed to an application-specific reduce function (e.g., summation of numeric values). From an implementation perspective, Hadoop job execution is orchestrated by a two-level hierarchy of control entities: the jobtracker, which runs on the Hadoop clusters master node and is the job-level control entity, and one or more tasktrackers, each running on every Hadoop compute node (termed slave).

B. *Shuffling in Hadoop*

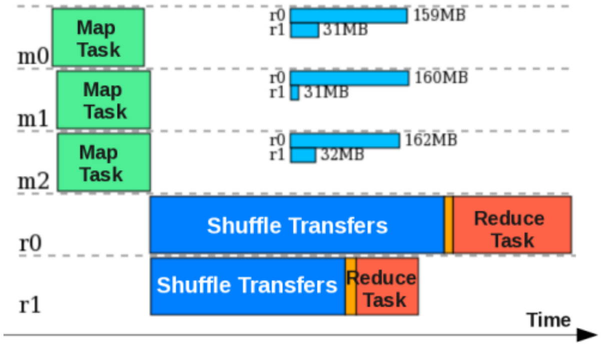

Fig. 3. Hadoop sort job sequence diagram

A recent analysis of MapReduce (MR) traces from Facebook revealed that 33% of the execution time of a large number of jobs is spent at the MapReduce phase that shuffles data between the various data-crunching nodes.

Figure. 3 depicts the sequence diagram of the execution of a sort job in a non-blocking network, obtained by a custom visualization tool. The job uses three map tasks (slots) and two reducers. For this job, the computing time for mappers and reducers consumes around 40% of total job completion time. It means that it almost takes 60% time in shuffling.

The main traffic occurs when mappers finish computing and start to transmit data to reducers. At this phase, lots of data might get into the network, causing congestion. In our project, we simulated a network framework in a fat-tree structure. Thus, there might exist several shortest path and how to choose the routing path remains a problem. Without SDN controllers, some switches might have to deal with much heavier traffic flow while the others have much lighter traffic. In such situation, the bandwidth can't be fully used. We want to have a balanced network to utilize the bandwidth resource as much as we can.

Since 60% time is consumed by network communication, potential optimization towards traffic could accelerate Hadoop job largely. The method is illustrated in detail in Section V.

V. Optimize Shuffle Traffic using SDN

A. *Introduction to SDN*

Conventional Internet framework is usually unable to be dynamically changed to keep up with application needs, which brings problems such as inflexibility. Software-Defined Networking allows network administrators to manage network services through abstraction of higher-level functionality. SDN offers fine-grained programmability of the network to deal with complex flow control.

SDN architectures decouple network control and forwarding functions, enabling network control to become directly programmable and the underlying infrastructure to be abstracted from applications and network services[].

OpenFlow is a mechanism which implements SDN concept. OpenFlow enables network controllers to determine the path of network packets across a network of switches. The controllers are distinct from the switches[26]. An OpenFlow switch consists of Secure Channel and Flow Table. Each time when there is a flow to send, it needs to be classified first by the Flow Table. After classification is done, the flow will be forwarded according to the rule set up by the controller. In our project, we used an OpenFlow controller called POX to implement all the controlling logic dealing with communication traffic.

B. *Network Scheduling*

In this section, we will introduce the details of network scheduling in SDN control module. In general, the network scheduling consists of two parts: routing and flow allocation algorithms. During startup, the scheduler gets a fat-tree network topology of the Hadoop cluster and compute a routing graph based on k-shortest paths algorithm (Dijkstra shortest-path algorithm). We choose to use k-shortest paths algorithm because we try to implement a multipath routing system which take use of more links and bandwidth in the network. The routing part defines how the packets or flows are transfered to the destination which is also called data plane. The k-shortest paths routing algorithm is hop-based and executed again to generate a updated routing graph only when the network topology has some changes.

In order to introduce the flow allocation algorithm we designed for our system, two things need to be explained: 1) the way to generate a $Load\ Graph$ with the information collected from each Hadoop nodes; 2) the details of flow allocation algorithm about how to utilize the $Load\ Graph$. Some formats of information are mentioned above. For example, the task activity $< inf\_type, node\_n, task\_header, timestamp >$ denotes that one node was distributed a new task or finished a task, where inf_type denotes the information type of this item,

$node\_n$ denotes the Hadoop node number, $task\_header$ contains the task details like task type and data size, $timestamp$ denotes the time when the information item is sent. With these information, we designed a $load\ weight\ algorithm$ to add/delete some load weight on switches. If the information shows that there is a new task distributed on a nodes, then the upwards switches are added some load weight. Formula are abstracted below: define a load weight w for this round of update, and w is related with the data size of this task

$$First - layer\ upwards\ swithces:\ weight(switch) += w$$
$$Second - layer\ upwards\ swithces:\ weight(switch) += w/2$$
$$Third - layer\ upwards\ swithces:\ weight(switch) += w/4$$

Because of the fat-tree topology, one layer may have lots of switches we just update first several switches on the routing table (based on k-shortest paths algorithm mentioned above). The strategy of load weight deletion is similar with the addition.

With the load graph, next step our network scheduler will generate new flow allocation rules for each switches and deploy them on switches regularly. For each switch, we contain a routing table which is computed by k-shortest paths algorithm. For each path, the flow or bandwidth available are in inverse proportion of the $load$ for the next hop. Therefore, we maintain a set of next hops for one destination, and allocate flow according to the $load$ of the next hops.

Thus, we design a network scheduling framework of routing computation and flow allocation. The last step it to deploy the new flow allocation rules on open source switches. We decide to deploy it regularly. More specifically, we deploy it after the $Load\ Graph$ is updated for several times.

Currently, all of the SDN network control logic of our system is implemented in the form of modular components within an OpenFlow controller POX. We will describe it in Section VI.

VI. Evaluation

A. Experimental Setup

The experimental setup can be concluded into three phases:

1) Setting up virtual Hadoop clusters: Since we have no access to physical computing cluster, we utilize virtual nodes for setting up Hadoop clusters. Vagrant[1] is a powerful tools used to eliminate repeat work by setting up, booting and provision all virtual nodes with Vagrantfile (/vagrant/Vagrantfile). We allocate 8GB memory to the namenode and 1GB to datanodes. Besides, We select ubuntu server 14.04 as the system for these nodes and only install the GUI on the name node for furthur Hadoop jobs tracking.

To set up Hadoop cluster on virtual nodes, we take the advantage of Cloudera Manager, which provide an integrated and fast way to install Hadoop. Different from conventional settings, we only activate the core Hadoop services like HDFS, YARN, etc. The main reason is that our virtual nodes are much less powerful than physical nodes and our benchmark is simple sorting job.

2) Setting up SDN network: For this phase, we use GNS3[2] to combine our virtual Hadoop clusters and Miniet. Briefly, GNS3 is a network simulator that can provide a totally isolated network environment, Figure 4 shows one of our examples in GNS3 for Hadoop cluster testing. Mininet is also a great way to develop, share, and experiment with OpenFlow and Software-Defined Networking systems. Through GNS3, we bind our virtual nodes as hosts in Miniet and we choose Fat-tree as the network topology.

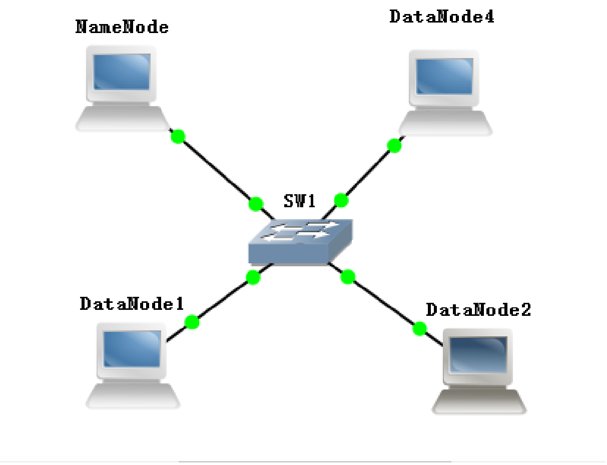

Fig. 4. GNS3 Sample

3) Parallelized Development: We also notice that the underlying parallelization of our development. That is, we isolate SDN algorithm development from Hadoop cluster installation and testing. Specifically, we feed the Mininet with Hadoop jobtrack logs available online and simulate the shuffle stage in Mininet host as precise as possible. By doing this, we can start developing SDN controller at the beginning of our project and don't have to wait for Hadoop clusters to be set up.

B. Experiments

In this work, we use sorting as our Hadoop job, because it is a simple and commonly used perfomance benchmark for Hadoop. And we use job complete time as our metric for evaluation. Our experiments mainly considering three factors that may affect the traffic optimization, controller algorithm, network scale and transfer scale. In detail, the number of nodes represents network scale while the number of transfers represents transfer scale.

1) Traffic optimization by different controller algorithm: First we validate our algorithm comparing to fixed switching and Pythia[6]. Table. I shows that our algorithm are significantly better than the baseline but slightly worse than Pythia, since Pythia use a more complex model which tries to predict the upcoming network flow.

Algorithm	Load Graph	Pythia	Fixed Switching
Job Complete Time	101.3	99.6	120.5
Optimized Ratio	15.9%	17.3%	0%

TABLE I. Optimized Ratio for Different Algorithms

[1] https://www.vagrantup.com/

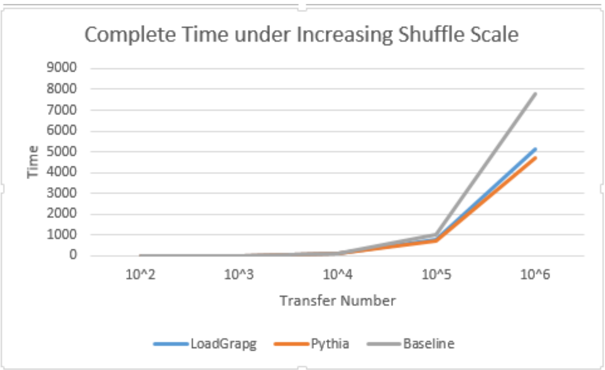

Fig. 5. Complete Time under Increasing Shuffle Scale (16 hosts)

2) Traffic optimization under different transfer scale: Then, we try to find wether data scale will affect the traffic optimization. We vary the number of transfer from 10^3 to 10^6. Figure .5 shows that bigger shuffle scale leads to high optimization ratio, which means our algorithms do help take better use of the network.

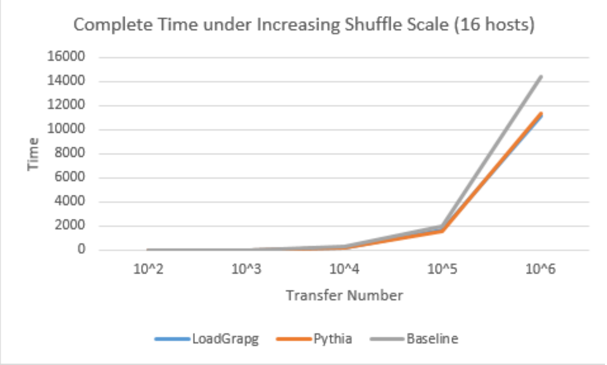

Fig. 6. Complete Time under Increasing Shuffle Scale (16 hosts)

3) Traffic Optimization under Different Network Scale: Last, we try to find wether network scale will affect the traffic optimization. We run the same experiment above in a 4-host network. Figure .6 shows that we achieve lower optimization ratio. The main reason is that there is much less path choice in a 4-host network than 16-host one, and thus less improvement can be make.

VII. Conclusion

We first point out the problem that shuffling take a large percentage of total running time of a Hadoop job, which due to insufficient usage of the network. Then, we implement a system based on Software-Define Network (SDN) to optimize network traffic for Hadoop shuffle stage. The idea is to build an SDN control module to adjust flow scheduling and achieve a more balanced flow allocation. The SDN controller collects the information from running mappers and reducers using an Auxiliary Process, and then updates a $Load\ Graph$ which reflects the workload in the network. The flow scheduling strategy working with k-shortest paths routing algorithm distributes different flows on the paths to the destination. We simulate the whole system with $Cloudera$, $Mininet$, $GNS3$, $OpenvSwitch$ and other softwares. By utilizing this system, we create a set of Hadoop jobs and observe the task completion time. Compared to the normal network, our system based on SDN improves the overal performance of Hadoop cluster system on big data and large clusters.

References

[1] G. Wang, T. Ng, and A. Shaikh, "Programming your network at run-time for big data applications," in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 103–108, ACM, 2012.

[2] M. Chowdhury and I. Stoica, "Coflow: a networking abstraction for cluster applications," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pp. 31–36, ACM, 2012.

[3] B. Sharma, R. Prabhakar, S.-H. Lim, M. T. Kandemir, and C. R. Das, "Mrorchestrator: A fine-grained resource orchestration framework for mapreduce clusters," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pp. 1–8, IEEE, 2012.

[4] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 98–109, 2011.

[5] A. Das, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and C. Yu, "Transparent and flexible network management for big data processing in the cloud," in *5th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud13)*, 2013.

[6] M. Veiga Neves, C. A. De Rose, K. Katrinis, and H. Franke, "Pythia: Faster big data in motion through predictive software-defined network optimization at runtime," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pp. 82–90, IEEE, 2014.

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

[2] http://www.gns3.com/