Tenant-Aware BigData Scheduling with Software-Defined Networking

I.Tasneem Akhthar
tasneem.akhthar@tecnico.ulisboa.pt

Instituto Superior Técnico, INESC-ID
Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal

Abstract. *The increase of data in the Internet world has created a need to process and acquire information from them using Big Data Analytics which in turn use data center for computing and storing purposes. The Big data analytics in data center needs a good network configuration to avoid delay or error in the network. but the traditional network could not avoid the error or dynamically create a network architecture. This gave rise to Software Defined Networking which configures, deploys and manages the network infrastructures and is most widely used in the data center network. Both these technology SDN and Big Data has benefited the data center network tremendously. SDN follows logically centralized approach, with which network allocation and scheduling can be performed with increased efficiency and reliability. Due to the emergence of cloud services it has becoming easy to access the computational resource. The client can rent large size of the computational resources in a very reasonable price, but unfortunately there is performance degradation in the network level as many clients would be using the computational instance through same communication channel and SLA (Service-level agreement) could not guarantee the performance in the network layer. This bring about the problem in the multi-tenant data center environment.Having these two technologies in mind we propose MR-FLOOD which is a conjugation of the Hadoop MapReduce framework and Floodlight controller. We brought these technologies together to give the tenants a fair bandwidth share and less latency using bandwidth or latency based job allocation strategy.Our assessments show that the above mentioned properties are achieved, being carried out in two common data center network topologies: Tree and Fat-tree.*

Keywords: Software-Defined Networking (SDN); Big-data processing; scheduling; multi-tenancy; cloud computing

1 Introduction

Massive increase in the usage of social networking, data intensive technologies, internet, cloud computing, has shown the growth of data in an unexpected extent. This huge data collected can be processed to extract helpful information, which is known as "Big data Analytics" and it is done by the data-intensive analytic framework such as Hadoop [17], Spark [19], Dryad [9] and many others.

From many years MapReduce is used as the big data framework which does data processing across large-scale computing clusters. Due to its simplicity and scalability it is used by big data analytics to parallelize the job and process the data. Among

many features, One of the main features of MapReduce is its ability to make use of data locality and minimize network transfers. But according to the research [7] there is performance degradation in MapReduce framework as there occurs bottleneck in the network due to job completion time in shuffle phase. For example, the recent study of MR traces from Facebook acknowlegde that the shuffle phase uses more than 50% of the job completion time [20]. This problem can be solved by optimizing the network used for communication, to reduce the response times.

MapReduce normally runs in large data centers (DCs) where network must be utilized in a proper way to increase the productivity. Many researchers have concentrated in optimizing the network based on the scheduling algorithms to improve the data locality and data transfer in the network. But very few work has been carried out in order to dynamically adapt the network behavior to MapReduce application's needs. So there must be some kind of controller which is already aware of the application's traffic demand in the network. These observations helped to improve the performance of individual MR applications and the overall network utilization, by introducing the revolutionary idea which is known as Software-Defined networking (SDN). SDN controls the network through software which in turn makes the network programmable. It has been introduced as a replacement for conventional networking methods so that it can meet today's market requirements.

Software-Defined Networking is a technique which allows a network controller to manage the network scheduling, in order to make full use of the resources and improve the performance of the software running on the network. In short, SDN decouples the network control (control plane) and forwarding functions (data plane) which enables the control over the network and abstracts the underlying network infrastructure for applications and network services.

Software-Defined Networking allows large scale networks to be coordinated and managed effectively from a logically centralized controller. Large scale cloud and networking systems are built leveraging the programmability and reconfigurability offered by SDN. For Big Data applications, software-Defined networks provide for the ability to program the network at runtime, in a manner such that, data movement is optimized for faster execution.

While existing MapReduce frameworks such as Hadoop are exploited for real time evaluation of QoS-driven service composition, SDN and OpenFlow controllers should be leveraged to enable a QoS-driven self-configuration of the MapReduce platforms. This research proposes a bandwidth and latency aware MapReduce service composition and self-configuration for big data Analytics with SDN, by extending and leveraging Hadoop MapReduce framework and floodlight SDN controller. Our project is also responsible for allowing the tenant to make use of the big data application(Hadoop) and still keep the data isolated from each tenant in the data center.

There are other bog data frameworks which is recently developed and they do give better performance but we chose MapReduce frameowrk as it was used in MRemu tool [12] to which we are upgrading in this project, we also improve the performance of MapReduce jobs through runtime communication using Software-Defined network, which is used to have a global view of the state of the data center network. This way we are able to allocate the MapReduce jobs in the Data center by ensuring the bandwidth and bandwidth. It is evaluated by trace-driven emulation-based experiments,

using popular benchmarks and real-world applications that are representative of significant MapReduce uses.

2 Related Work

In the previous section we provided an insight on SDN, now we discuss about OpenFlow protocol as SDN uses it. The OpenFlow architecture has three important concepts: the network consist of OpenFlow switches, which becomes the data plane; the control plane is made by at least one OpenFlow controller; and there is a secure channel between the switch and the control plane. Each and every switch is a "dumb" forwarding device, that simply uses its flow table to determine where the packet has to be transmitted. A flow table is composed by a series of flow entries. Each flow entry has the header fields on which it will try to match incoming packets (e.g. Ethernet destination address), actions to perform when an incoming packet matches this table entry (e.g. forward to a specified port or flood to all ports), and counters that hold statistical information about each flow (e.g. number of packets and bytes transmitted in this flow). When an incoming packet does not match any entry on the flow table, it is sent to the controller using the secure channel, which will decide what to do with this packet.

2.1 BigData Application Using OpenFlow

There are many number of techniques which uses OpenFlow to improve the transfer of data to the nodes with better bandwidth utilization. Some of them are discussed below:

Zhao li et. al. [10] proposed OFScheduler which is a network optimizer based on OpenFlow for improving MapReduce operations in a heterogeneous cluster. OFscheduler is aimed to increase the utilization of bandwidth and balance the workload of the links in the network. The author uses tag to identify different types of traffic and then OpenFlow tailors and transmits the flow dynamically.

FlowComb is a network management framework [5], which is also based on OpenFlow. It benefits Hadoop with increased in bandwidth utilization and decreased data execution times. The centralized decision engine present in FlowComb, collects data from software installed on application servers and decides, how to schedule forthcoming flows globally without congesting the network.

Qin et al., [14] have proposed a framework that employs SDN in Hadoop to decrease the time taken by data to reach the distributed data nodes from the mappers. To avoid the delay the author has developed a technique which eventually measures the current available bandwidth on the links through OpenFlow protocol and schedules tasks in a way which decreases the execution time. He named the proposed technique as 'bandwidth aware task scheduler' (BASS) which utilized SDN and it manages the available bandwidth of the links, and then allots into the time slot(TS), later BASS decides to allocate the task remotely or locally with respect to the completion time. The main plus point in BASS is that it does task scheduling, even when the OpenFlow controller has scanty amount of bandwidth to consider.

Sandhaya Narayan et al., [11] uses OpenFlow to control the network resources in Hadoop. The most traffic generating phase in Hadoop is Shuffle phase where the intermediate data is moved from Mappers to the Reducers. Due to the heavy traffic in the

Shuffle phase, which in turn makes the bandwidth link unavailable between Mappers and Reducers, causes delay in the Reducer. Consequently lowering the performance of the Hadoop cluster. So the author has used OpenFlow technology which is used in SDN to furnish better link bandwidth for shuffle traffic which simultaneously decreases the Hadoop job's execution time.

Wang et al., [15] have proposed a application-aware networking setup based on SDN controller with optical switching, in which the network configuration is made easier by using the integrated control plane in job placement. The job is scheduled in two ways *i)Bin packaging placement ii)Batch processing*. Due to application awareness of the network there is improvement in the big data performance by allocating and scheduling bandwidth, as well as the completion time of the job is minimized drastically. The author used Hadoop as an example of the big data application framework. The challenge which the authors faced during configuration of the network for Hadoop task was to handle the aggregated traffic in the mappers and reducers, which was resolved by ranking the racks based on the traffic demands and then the racks needed to be added in the tree topology in descending order of the demand.

Freguson et al., [6] has developed an API named PANE which is implemented on OpenFlow Controller that gives read and write authority from the network administrator to end user. With the help of PANE,the end users are allowed to work in the network , which means that the end user and their application can manage the network according to their need. It allows applications to contact the controller to request for resources or set up rules for future traffic. PANE solves two issues (i)It allows multiple end user (principal) to control the network resources (ii) It successfully solves the conflict between the end users requests, meanwhile it also maintains the level of fairness and security.

End Users (principal) in PANE which gets read and write authority from the network administrator can issue three type of message such as Queries,Hints, Request [6]. These three message are used by principal when the user wants to request for bandwidth or access control resources(Request Message),when user wants to learn about the traffic between host or available bandwidth (Query), whereas the last message notifies the end user about the current or future traffic characteristics.

Participatory Network restricts on the limits of the end user authority with the help of *shares*. Share mentions which end user can send which kind of message in the flowgroup, where the flowgroup refers to the messages in some of the network flow. A share has three segements: *principals, privileges and flowgroup* [6]. The authors demonstrate the feasibility of their proposed approach with four distributed applications such as Ekiga, SSHGuard, Zookeeper and Hadoop.

Xiaolin et al., proposed Palantir [18], which is a SDN service specific for distributed frameworks. It was build to abstract proximity information from the network for the distributed frameworks such as MapReduce, Dryad. Earlier the network administrator has to manually configure the network for the distributed framework but Palantir offered great help to the network administrator by defining and capturing the proximity information. To implement Locality Aware Task Scheduling and Rack Aware replica placement we need to observe the network proximity information, where Palantir allowed the distributed framework to express their definition of proximity. With the help of this definition the Palantir automatically abstracted the proximity graph from phys-

ical network topology. Palantir is leveraged on FloodLight controller [1] where it uses the existing device and topology manager of FloodLight to abstract the network information. Palantir had some concerns on scalability of the system so they tried to reduce the computational cost by caching the abstracted proximity domain for each registered framework in memory. When the framework corresponding to the Palantir was shutdown or unregistered then the cached results are deleted from the memory.

3 *MR-Flood*

We now describe our solution *MR-Flood*. We start this chapter by providing a use case for our system, in the upcoming paragraph. Then, in section 3.1 we provide a detailed architecture explanation, mainly on the interactions between the component of our thesis solution. In section 3.2 we go in-depth to the components presented in the previous section, and describe the internal working of each one. Then, in section 3.3, we summarize this chapter.

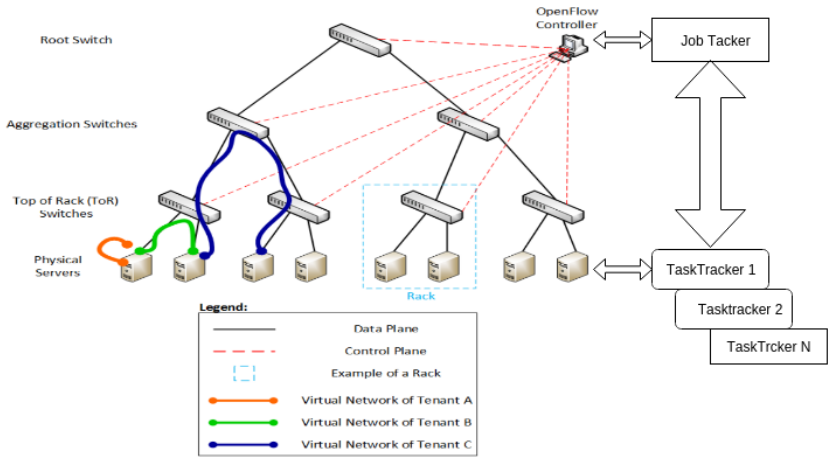

Fig. 1: High level architecture of the solution with a simple data center tree topology.

In Figure 1 we show the high level architecture of our solution, which is inspired by ViTena [4] in this use case we are also using a simple tree topology with depth equal to 3 and fanout equal to 2. Most of our work will be similar to ViTena [4] but with the changes due to the inclusion of MapReduce job scheduling in the VMs which is allocated as described in ViTena [4]. Here we are using the traditional topology to have a better understanding of our architecture. The OpenFlow controller is the main component of our solution, as it is responsible for running the "Latency and bandwidth aware job scheduling" algorithm in order to map the requests on the physical network, and then dynamically program the switches to deploy the requested task. We also introduce MapReduce emulator (rightmost) which is another component of this solution and it consist of Job Tracker and Task Tracker. It emulates as MapReduce framework.

[1] http://www. projectfloodlight. org/floodlight/

Now we explain about the Openflow switch where all the switches acts as a packet forwarder as they do not have any intelligence to forward the packet, it listens only to the controller and acts according to the rule described by the controller. The controller is logically connected with all the switches in the network, which is depicted by a dashed red lines that becomes the control plane. They are only logically separated but not physically separated from each other.

Our algorithm "Latency and bandwidth aware job scheduling" tries to schedule the batch of task on the smallest available subset of the physical network (on the same physical server, then on the same rack, and so on). Our aim to maximize the proximity of Batch of task in the VMs belonging to the same tenant, which results in minimizing the number of hops between those VMs, in turn it reduces the latency and having the task placed in proximity i.e to be in the same server or in different rack, this saves the bandwidth usage in the upper links of the tree, which helps in saving the bandwidth in the upper links as the network is scarcer in data center [3]. By this we can allocate more number of task without making the network links to become bottleneck.

Now we explain you the Figure 1, where we show you an example of three virtual network placements according to algorithm presented in ViTena [4]. The virtual network of tenant A represents the best case possible where all the batch of task can be mapped on the same physical server. In this, there is no usage of the network due to which bandwidth is saved for future requests of the task allocation. Now in other situation where the batch of task is split into two cases based on latency and bandwidth oriented request such that the given batch of task cannot be mapped into the same server. For instance let us assume that the task is latency oriented and the request cannot be mapped into a single server then we try to allocate the request in other servers belonging to the same rack as shown in the virtual network of tenant B this drastically reduces the latency. In the second case we assume that the request is bandwidth oriented then our algorithm looks for the server which are scattered apart to allocate the request (i.e. not in the same rack) as shown in the virtual network of tenant C to ensure that bandwidth and latency are not compromised.And the nearest free server are kept for the future task which are latency oriented.

Our algorithm ensures that the network, can provide the bandwidth and latency guarantees requested by each Batch of task Apart from this we also use the centralized information in the controller to provide two properties not seen in the systems we have surveyed, those are the fair bandwidth sharing among tasks and reduction of latency for the Batch of Task requests. Fair bandwidth sharing is achieved by instructing every switch used by a virtual network to follow the QoS disciplines by creating a new queue for that task. Secondly the reduction of latency is achieved by ensuring latency and bandwidth for batch of task algorithm, by placing the task using best fit strategy. To know the heuristics we need to abstract the current network from the OpenFlow controller.

3.1 Architecture of the Proposed Solution

Now we describe in greater detail about the architecture shown in the previous section. Figure 2 is the similar to the previous architecture but here it shows in greater detail about the component of our project. Thus, in Figure 2, it contains the software that will be used for the execution of our alogorithm.

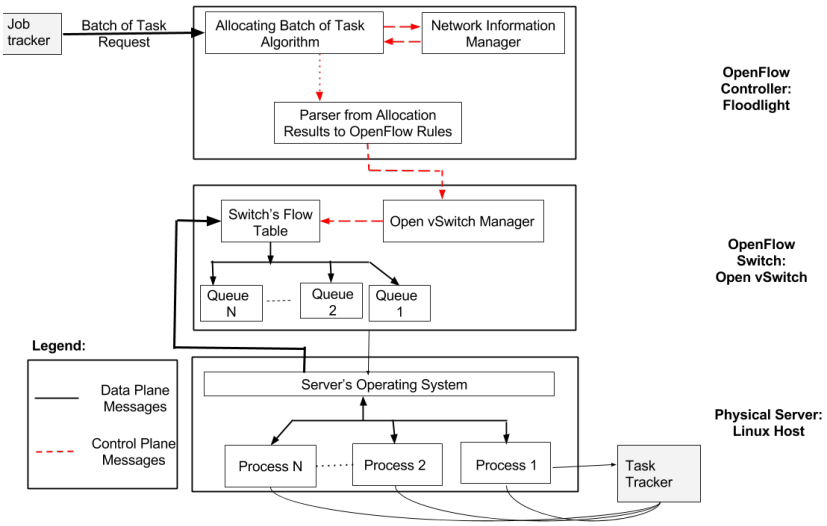

Fig. 2: Software architecture of the components present in the solution.

Let us justify our choices regarding the components shown in Figure 2. For the OpenFlow controller, we will use Floodlight for two reasons: first because it is based in Java, which as stated earlier gives the highest performance; and second because it is build using Apache Ant,which is very easy and flexible in use. Regarding the other components, Switch and Physical Server, the choices are Open vSwitch and Linux Host (respectively) because this solution will be implemented within the Mininet emulator, that uses those components. We will use Mininet because it is open-source and also because it is used by the MRemu tool [12]. Our solution can be executed in the real data center network with or without any changes. The only exception is the Linux Host, where there would be a hypervisor running and it would manage the VMs inside the host machine. We make an assumption to simplify our solution that all physical server have the same CPU frequency so the tenant asks for the percentage of CPU instead of the CPU frequency.

Now we depict the flow of information in the system which uses our algorithm. First of all, the BigData application's task expresses its demands in a Batch of task request (e.g. an JSON file). This consists in defining two things: the number of VMs required (and also the percentage of CPU of each one) as well as the bandwidth required between the VMs that will be connected (expressed in MBit/s). The batch of task request is fed into the ensuring latency and bandwidth for batch of task algorithm (detailed in the next section), that is running in the OpenFlow controller. Upon receiving the request, the algorithm contacts the network information manager to get the current state of the network. Based on this state, the algorithm determines (if the request is accepted) where this batch of task will be allocated. One important thing we should know is, all the requests are processed by the controller, this updated view of the network involves zero control messages over the network, since the controller just has to update this information when it processes a new task allocation request.

The controller then translates the result of the algorithm (i.e. the affected switches and hosts) to OpenFlow rule(s) to reprogram the switch(es). Upon receiving this message, the Open vSwitch manager takes two actions: creates a new queue for the batch of task, with the minimum bandwidth present in the received message; and installs a new rule in the switch's flow table to forward packets from that batch of task to the newly created queue. This means that there will be one queue for each batch of task passing on that switch. In this way, a queue (Batch Of Task) can use more bandwidth than its minimum when the other queues are not using it.

The bandwidth share between queues is made fairly according to the minimum bandwidth a queue has. Thus, the resource usage is maximized, as the tenants share unused bandwidth in the fair manner, and they also get their minimum bandwidth guarantee when the network is saturated. Suppose if the network is saturated, packets may be dropped at the switch. This ensures performance isolation at the network level. As the Open vSwitch is based on the Linux kernel, we will use the traffic control (tc) utility HTB (Hierarchy Token Bucket) to make the configuration of the queues.

We will represent and implement Batch of task as processes. To simulate tenants workloads, each process will be running a traffic generator. As depicted in Figure 2, upon the necessary configurations, each process (i.e. Batch of Task) can communicate with other processes on the same virtual network, using either the operating system (in case the processes are on the same server), or contacting its adjacent switch which will use the flow table to check to which queue it should forward this solicitation (in case the processes are on different servers). To make this distinction, each physical server needs to keep a list of the processes (and the corresponding virtual network of each one) it owns.

4 Evaluation Methodology

The traces which we are using to generate traffic in our experiment were obtained from the MRemu github repository with modification explained in previous section. Neves et al. [12] used the traces which was obtained from the HiBench benchmark suite. [8]

- **WordCount** - This is the most common application used during the MapReduce evalution. It has a text input data where each words are counted when they occur. The traces had 50 GB of input data.
- **TeraSort** - It is an application where huge data is sorted to check the time it takes to sort using the MapReduce framework. With the help of this application the MapReduce framework is evaluated. we use the traces of 31 GB of data from HiBench benchmark suite.
- **Sort** - We also use the Sorting application where it sorts the input data which is in text. This input data is generated from a Random TextWriter. This application is widely used to evaluate the performance factor. In this application 32 GB of data was sorted to obtain Job traces.
- **PageRank** - This application is implemented using the page-rank [8] algorithm, where it calculates the ranks if the web page by counting the number of reference links in the web page. So this application is also used in our evaluation strategy. Here we use 500k pages to process the page rank which was configured by Neves

et al. This 500k pages of data has 1 GB of input data where Neves et al. used Pegasus project [2] to derive the job traces. Hence, this is also used by us to know the efficiency of the MapReduce framework during job scheduling.
- **Nutch** - Apache Nutch [1] is a web crawler, used for page indexing purposes, this is a;so used in MapReduce framework and Neves et al [12] used 5M of pages to do the indexing and it was approximately 8 GB of data.
- **Bayesian Classification** - This is last application which we are using for our evaluation strategy. It is based on the Naive Bayesian [8] algorithm. It is basically used in Data mining, Machine learning, Knowledge discovery and it is part of Apache Mahout [13]. For this application Neves [12] configured 100K pages to be used as a job trace.

MaxiNet [16] is best to emulate the data center networks as it gives the realistic feel of the data center networks and it was our choice at first. But then, it was not supported by Mremu [12] framework. so we have to use Mininet to create our data center network topologies.

We use two different types of mininet topologies developed by [4] to evaluate our experiment. These two topologies were used to emulate the data center.

- **Tree topology** - It has 125 servers, 31 switches and 155 links. This consists a Tree topology with a depth equal to 3 and a fanout equal to 5.
- **Fat-tree topology** - It has 128 servers, 160 switches and 384 links in the network. Here the k factor is set to 32, which means each switch has 32 ports.

4.1 Goal - Job Completion time

In this goal we want to evaluate the job completion time using tree topology. The job traces are serially processed based on the latency/bandwidth parameter. We evaluate to check how efficient our experiment is when compared with the Zhuoyao Zhang et al., [21] where he has not used SDN to abstract the underlying information of the data center network.

4.1.1 Tree Topology In Figure 3, we can see the results of obtained with a Tree topology:

It has a row of blue bar which depicts the Job completion time when the latency was set as parameter and the red bar in the chart depicts the Job completion time when bandwidth was set as parameter for our job scheduling algorithm.

Our results for the job completion time are better when compared with [21]. Our job scheduling algorithm processed the request of all jobs was quickly and achieved 47.3% less time when compared to Zhuoyao Zhang et al., [21] work.

By analyzing Figure 3, we can see that bandwidth oriented task has higher job completion time. This is because the bandwidth task are scattered into the servers. And the latency based task are closely kept in the server due to which the time taken to complete the job is lower when compared to the bandwidth based job.

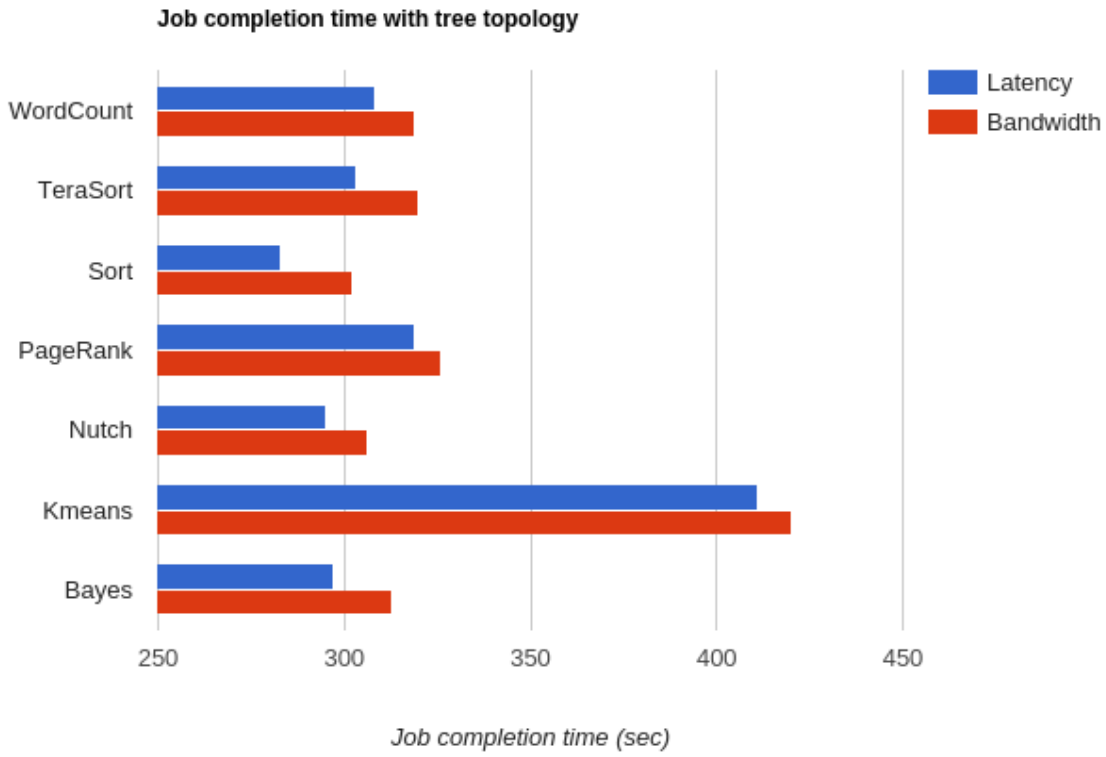

Fig. 3: Job completion time using a Tree topology.

4.1.2 Fat-tree Topology The results for Job completion time using a Fat-tree topology, are presented in Figure 4:

In this topology we have more links and switches as the k factor is set to 32 and due to this, the fat tree topology would have more path than the Tree topology.

As described in the previous topology the blue bar depicts the job completion time of the latency oriented task and the red bar in the chart depicts the job completion time of bandwidth oriented task.

Looking at Figure 4, we can see that the completion time for all the task using this topology is higher than the one using a Tree topology. And when this result was compared with Zhuoyao Zhang et al., [21] work, it showed a variation of 48.1% which seems to be better. Fat tree took more time to complete the job completion time when compared to tree topology as it had more number of paths between two nodes.

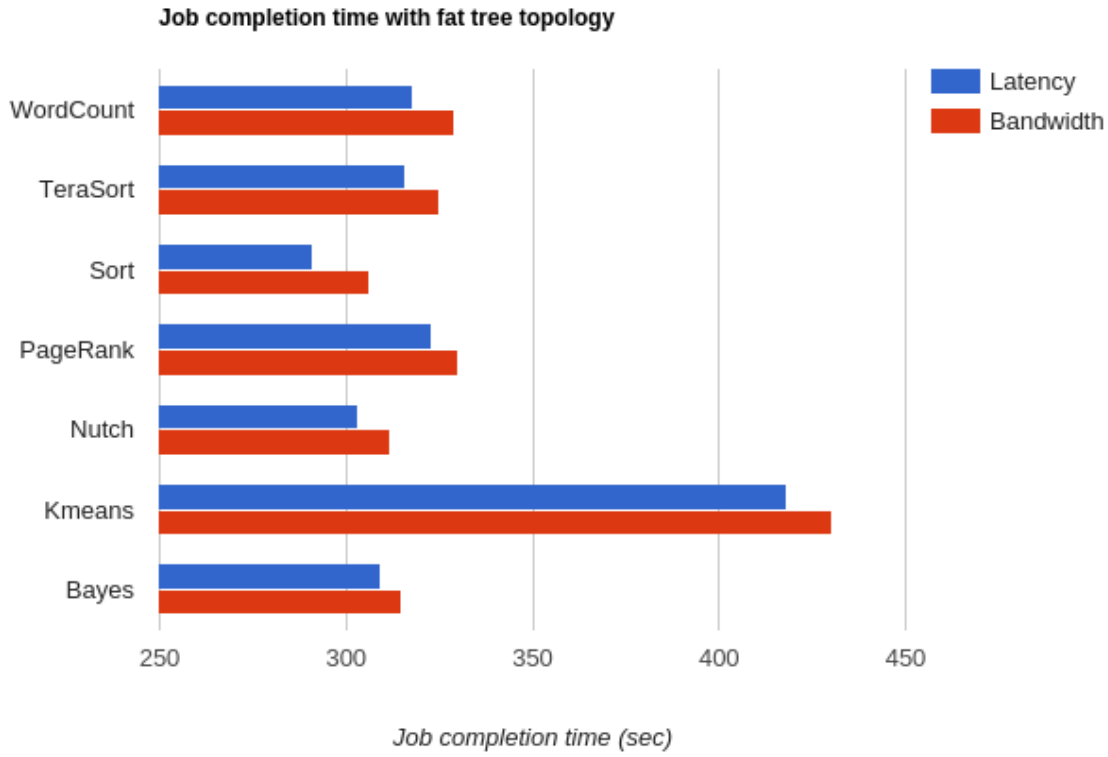

Fig. 4: Job completion time using fat tree topology.

5 Conclusion

we have addressed the topic of Job scheduling, network management and configuration in Data Center networks. We know that current Data Centers lack network performance guarantees, since all tenants interchangeably share the network. This makes

the performance to degrade in a tenant's application, since the tenant dependents on factors outside of its control.

These problems are solved using the abstraction of underlying networks. Using the abstraction we gathered the network parameters such as bandwidth, latency and CPU to provide better resource to the tenant without degrading the performance. All this was possible by using the Floodlight SDN controller.

We designed an OpenFlow controller that is able to allocate the batch of task (with bandwidth guarantees) in a work-conservative system, achieving high consolidation on the allocation of tasks and high resource utilization of the Data Center's resources. We have designed our task allocation algorithm taking into account the environment where it will run in, a Data Center, which means that it has to scale well.

With our solution well defined, we have implemented our controller on top of the Floodlight open-source project with the Hadoop Mapreduce emulator. We began studying both the tools so that we can implement our logic into the floodlight controller with which our job scheduling would fulfill its demand of allocating the batch of task. But for this we need to develop a new module inside the Floodlight controller. Apart from that we also make modification inside the MapReduce emulator to be able to provide tenant aware job scheduling.

In MapReduce emulator we have overcome the limitation Neves et al, had. We have enhanced the tool by introducing "Concurrent job execution" And we also made many changes so that we can make our implementation possible.

After the comprehensive evaluation of our implementation. we use the developed scripts [4] to create two Data Center network topologies: Tree and Fat-tree. After the evaluation conducted the results were outstanding, which shows that we have pretty much accomplished the goals we have set out in the beginning. From the evaluation, we can see that our system has: low execution time, high consolidation of task allocation and high resource utilization.

References

1. Apache nutch.
2. Pegasus: A peta-scale graph mining system implementation and observations.
3. Kashif Bilal, Samee Ullah Khan, Joanna Kolodziej, Limin Zhang, Khizar Hayat, Sajjad Ahmad Madani, Nasro Min-Allah, Lizhe Wang, and Dan Chen. A comparative study of data center network architectures. In *ECMS*, pages 526–532, 2012.
4. Daniel Caixinha, Pradeeban Kathiravelu, and Luis Veiga. Vitena: An sdn-based virtua; network embedding algorithm for multi-tenant data centers., In 15th IEEE International Symposium on Network Computing and Applications(NCA 2016).
5. Anupam Das, Cristian Lumezanu, Yueping Zhang, Vishal Singh, Guofei Jiang, and Curtis Yu. Transparent and flexible network management for big data processing in the cloud. In *5th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'13)*, 2013.
6. Andrew D Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca, and Shriram Krishnamurthi. Participatory networking: An api for application control of sdns. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 327–338. ACM, 2013.
7. Mohammad Hammoud, M Suhail Rehman, and Majd F Sakr. Center-of-gravity reduce task scheduling to lower mapreduce network traffic. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 49–58. IEEE, 2012.

8. Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *New Frontiers in Information and Software as Services*, pages 209–228. Springer, 2011.
9. Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 59–72. ACM, 2007.
10. Zhao Li, Yao Shen, Bin Yao, and Minyi Guo. Ofscheduler: a dynamic network optimizer for mapreduce in heterogeneous cluster. *International Journal of Parallel Programming*, 43(3):472–488, 2015.
11. Sandhya Narayan, Stuart Bailey, and Anand Daga. Hadoop acceleration in an openflow-based cluster. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis, Salt Lake City, UT, USA, November 10-16, 2012*, pages 535–538. IEEE Computer Society, 2012.
12. Marcelo Veiga Neves, Cesar AF De Rose, and Kostas Katrinis. Mremu: An emulation-based framework for datacenter network experimentation using realistic mapreduce traffic. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on*, pages 174–177. IEEE, 2015.
13. Sean Owen, Robin Anil, Ted Dunning, and Ellen Friedman. Mahout in action. 2012.
14. Peng Qin, Bin Dai, Benxiong Huang, and Guan Xu. Bandwidth-aware scheduling with sdn in hadoop: A new trend for big data. *arXiv preprint arXiv:1403.2800*, 2014.
15. Guohui Wang, TS Ng, and Anees Shaikh. Programming your network at run-time for big data applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 103–108. ACM, 2012.
16. Philip Wette, Martin Draxler, Arne Schwabe, Felix Wallaschek, Mohammad Hassan Zahraee, and Holger Karl. Maxinet: Distributed emulation of software-defined networks. In *Networking Conference, 2014 IFIP*, pages 1–9. IEEE, 2014.
17. Tom White. *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.
18. Ze Yu, Min Li, Xin Yang, and Xiaolin Li. Palantir: Reseizing network proximity in large-scale distributed computing frameworks using sdn. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 440–447. IEEE, 2014.
19. Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, page 10, 2010.
20. Deze Zeng, Lin Gu, and Song Guo. *Cloud Networking for Big Data*. Springer, 2015.
21. Zhuoyao Zhang, Ludmila Cherkasova, and Boon Thau Loo. Optimizing cost and performance trade-offs for mapreduce job processing in the cloud. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–8. IEEE, 2014.