Optimizing Task Scheduling for HPC using Software Defined Network

Vidya Chitre, Deven Shah

Abstract*: Features of cloud computing in terms of reliability, scalability, and resource pooling have inveigled scientists to deploy high performance computing (HPC) applications on cloud. Nevertheless, HPC applications have major challenges on cloud that emasculate gained benefits. This study aims to analyze challenges that are faced by HPC applications on cloud and focus on how the performance of cloud can be increased through a platform. In this paper, different approaches are discussed for improving HPC task[1] scheduling performance on cloud using software-defined networking. Moreover, a task scheduling architecture is proposed to manage task scheduling of HPC applications, which can be later on used as a service to improve the profit of service providers. Bandwidth and capacity of virtual machines are the main parameters that are considered.

***Index Terms*: Software Defined Network, Task Scheduling, HPCaaS, Mininet, Virtualization**

I. INTRODUCTION

A characteristic of cloud computing (CC) that is time and again not considered crucial but rather as an unexpected result of resource pooling involves multitenancy. High performance computing (HPC) applications usually run with direct physical access to hardware thereby offering paramount performance. Consequently, traditional HPC platforms are tremendously perfected to obtain best performance of parallel processing. RedeployingHPC applications straightforwardly to cloud can cause performance penalty and snuff out the advantages gained from cloud services[1].

Software-defined networking (SDN) is an emerging network architecture that splendidly fits with the network virtualization of cloud [2]. The unique architectural design of SDN offers programmability that decouples network intelligence from data plane and has controller plane as a separate layer. SDN's controller plane possesses complete network information and decides network rules and routing policies to improve performance by scheduling HPC tasks on cloud.

This study aims to analyze and improve the performance of HPC applications through SDN-enabled task scheduling algorithm. The proposed algorithm performs better as the number of shared tenants increase thereby maximizing benefits through resource sharing between additional tenants and not only improving resource utilization but also improving scalability of HPC applications for HPC users.

Revised Manuscript Received on May 25, 2019.

Vidya Chitre, Dr. Deven Shah
Research Scholar, Department of Information Technology,Terna Engineering College, Mumbai, India, Research Guide,Department of Information Technology, Thakur College of Engineering and Technology, Mumbai, India

The remainder of this paper is organized as follows. Section II discusses about high-performance-computing-as-a-service (HPCaaS) and SDN. Section III describes related work in task scheduling for HPCaaS using SDN. Section IV illustrates about the proposed methodology. Section V describes implementation strategy of the proposed system. Section VI concludes the study.

II. RELATED WORK

A. High-performance-computing-as-a-service (HPCaaS)

Most of the HPC applications require high computational capacity as well as selection of careful network consideration. Cloud offers computing services such as processing power, storage, and network bandwidth to end users. CC is a technology that offers on-demand access to a pool of configurable computing resources. Its major characteristics include on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service [3]. Cloud offers a variety of services such as software-as-a-service (SaaS), platform-as-a-service (PaaS), and infrastructure-as-a-service (IaaS). In addition to these, HPCaaS is an upcoming service in this field [4]. Cloud intends to provide computational capacity by offering a number of resources as a service. Deploying HPC applications on cloud and using cloud infrastructures for HPC programs is known as HPCaaS. Note that primary motivations for deploying HPC applications on cloud are coordinated with typical benefits of CC such as resource utilization, cost efficiency, and flexibility.

B. Benefits of HPCaaS

Cloud possesses the capability to offer countless unique advantages for HPC users in addition to the general benefits of CC [4].

Easy maintenance and administration

Most of the HPC users are scientists and physicians who do not have sufficient amount of computer background. As a result, they give importance to ready running service and maintenance of an HPC cluster.

Resource utilization

Cloud offers a dynamic and scalable infrastructure for applications, and depending on the demand of an application for business, it can be dynamically scaled up and down. According to a survey, HPC users do not require resources for a long time period. Occasionally, there could be a possibility for huge and sudden resource

requirement. As a result, requirement of cloud-based resources and its utilization shall benefit on-demand pay-per-use model thereby converting capital expenses to operational expenses.

C. HPCaaS-based challenges on cloud

Several research works compared the performance of HPC applications on cloud with on-premise infrastructure and concluded that today's cloud cannot contend with supercomputers because of the following reasons [5].

Multitenancy

It is an essential characteristic of cloud that allows cloud providers to share resources amongst multiple tenants. Cloud providers can maximize benefits by increasing the degree of multitenancy. However, HPC applications demand direct access to dedicated hardware using batch scheduling, while shared resources on cloud complicate the performance of HPC applications. Resource sharing degrades performance as same resources are utilized by different applications.

Virtualization overhead

Virtualization in hypervisor includes unwanted overhead by adding a software layer and preventing applications to have direct access to hardware resources. Note that virtualization overhead may be different for different hardware. Because of hardware support, virtualization overhead for processors is remarkably less than network virtualization overhead [6].

Network bandwidth and latency

Network interconnects resource sharing in cloud that is shared by several tenants. Therefore, bandwidth may keep changing and network latency may be unpredictable. The bandwidth that is obtained in many cases is much less than expected.

D. SDN

It is an emerging network architecture that not only decouples network control but also packet flow in data plane [1, 7].

The proposed method makes network management adaptable for high bandwidth and dynamic nature of today's highly scalable applications. Centralized programmable control plane is permitted by network technology to manage entire data plane.It allows open application programming interface (API) communication between hardware and operating system (OS) and network elements and OS. However, SDN integration with cloud management software, i.e., OpenStack, has not been investigated.

III. LITERATURE SURVEY

Of late, HPC users have joined large community of cloud users by deploying applications on cloud. However, standard clouds dissatisfyunique HPC application requirements such as batch processing, access to hardware, bypassing OS kernel, and high-performance execution. Research about HPC on cloud can be divided into two groups.

- Performance analysis
- Performance improvement

A. Performance analysis

Leite et al. [8] proposed a cloud architecture—Excalibur—running parallel applications. It involves a distributed file system, a job scheduler, and workflow management alongside other components. The proposed framework minimizes data movement to reduce cost and execution time and adjusts workloads to minimize job makespan. In addition, the architecture possesses the capability of expanding dynamically by inserting more instances in an autonomic manner. Results of autonomic configuration of the proposed architecture revealed 73% and 84% reduction in execution time and cost, respectively.Gupta et al. [9] carried out extensive work for evaluating cloud architecture for HPC applications to reduce cost. Eucalyptus and Open Cirrus were used as cloud testbeds, and their performance and cost were compared with a physical traditional HPC-optimized cluster. Their study revealed that Open Cirrus exhibited better performance than Eucalyptus. However, cloud environments were not able to overcome the performance of physical cluster when there is an increase in the number of cores and size of applications. Analysis in terms of cost and performance indicated that cloud is more cost-effective for parallel applications that do not require high network communications. Kim and Feamster [10] discussed about HPC performance evaluation on cloud that considers an application's turnaround time and cost. They compared Amazon EC2 public cloud with HPC clusters at Lawrence Livermore National Laboratory. Results of their study proved to be promising for cloud as the turnaround time for some applications was less than that for physical clusters. Conversely, as anticipated, HPC cluster's raw performance was superior to cloud.According to a survey, the two main challenges for HPCaaS identified by various studies are networking and virtualization overheads.

B. Performance improvement

AbdelBaky et al. [11] introduced a framework to transform a supercomputer into cloud that supports HPC resource accessibilitythrough IaaS, PaaS, and SaaS abstractions. They were of the opinion that efforts to provide HPC resources for scientific applications in the form of HPC in cloud, HPC plus cloud, and HPC as a cloud have not reached completely caused by underlying commodity hardware's limited capabilities, lack of high-speed interconnections, and physical or geographical distance between servers and virtualization overhead. Furthermore, their study identified ease of use, elasticity, and accessibility of cloud as the primary benefits of HPCaaS.Taifi et al. [12] proposed a paradigm of HPCaaS architecture for developing HPC clusters over a private cloud. High-throughput connections were used, and an Ethernet network was connected. Infinity band can be used for connection of VM, and use of computing nodes can offer better performance. These high bandwidth interconnects personate an imperative part in convalescing network performance, which is necessitous for HPC applications. In addition,

their study identified three foremost benefits for HPCaaS, namely resource flexibility, efficiency, and cost reduction. Nevertheless, to achieve the abovementioned advantages, challenges such as virtualization overhead, administrative complexity, and a programming model need to be dealt with.

IV. TASK SCHEDULER ON CLOUD

Multitenant cloud network gives rise to erratic and unplanned bandwidths with extra average latency. Even though controlling rapid changes in bandwidth is not easy without dedicating a link to a tenant, it is effortless to monitor changes using SDN controller's API and take necessary actions. Thus, to be aware of current system bandwidth, a task scheduling system is proposed that benefits from SDN technology.Figure 1 shows the conceptual architecture for SDN-enabled task scheduler. The job scheduler is responsible for dividing HPC jobs into several tasks and placing them in queue for task scheduler. Tasks comprising metadata of required data from shared file system are allocated to a suitable VM worker by task scheduler that takes advantage of SDN controller to be aware of all bandwidth links. Note that a task scheduling algorithm runs within a task scheduler and not within a job scheduler.

The queue is split into two parts:

- Running state: A state where a task queue is full and the task is ready to execute; so, VM is launched.
- Waiting state: A state in which VM is not launched.

For a switch's empty port, VM is considered to be in waiting state.

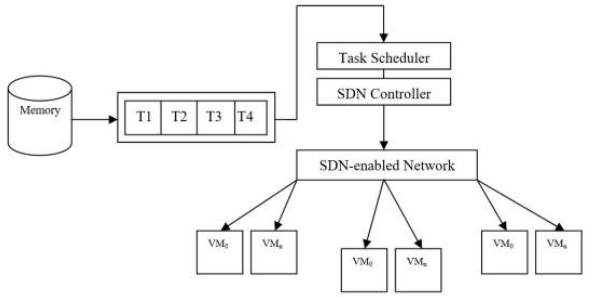


Fig. 1. Conceptual architecture for SDN-enabled task scheduler

A. SDN-enabled task scheduling algorithm

This algorithm runs within a task scheduler in our proposed system. As the controller is always aware of network properties, it is connected to a load balancer. For VMs that are not yet launched, the next available idle time (Ti) will be same as the that is time needed for them to launch and become ready (Ri). The task scheduler can calculate a task j's finishing time on all VMs and select only that task that has minimum value.

∀i =1 to N, ∀j = 1 to M;
Time j = MIN;
i = 1 to N (Ti + DTij + Rij);

Where

N = number of VMs,
M = number of tasks,
Time j = Duration of the completion time of task j,
D = Size of data (bytes)
SDN-enabled task scheduling algorithm:

- Determine data transfer time for data of task j to VM I (duration time)
- Calculate task j'sfinishing time on all VMs and select only that task that has minimum value.
- Assign the selected task to VM
- If job queue is full, then launch VM

else
terminate VM

V. IMPLEMENTATION

The proposed system is implemented on a network emulator Mininet. Based on infrastructure and platforms, Mininet is deployed on cloud that is integrated with OpenDaylight—a popular community-led and industry supported open-source SDN controller framework that accelerates adoption of SDN and network function virtualization[13]. The OpenDaylight SDN controller collects information about available links and provides it to a task scheduler. Next, the task scheduler uses those links to determine suitable paths and idle machines to work on.

A. Mininet SDN platform

There are countless simulation or emulation platforms that can be used to scrutinize and estimate the concept of SDN in virtual environments. Mininet emulator is believed to be one of the most well-liked and competent SDN-based OpenFlow platform because of its integrity, flexibility, availability, and simplicity in developing SDN networks. Note that such networks include OpenFlow switches, controllers, and hosts that can be connected either through Ethernet or secure interfaces based on secure shell (SSH) protocol in a single Linux kernel. Mininet makes use of Ubuntu Linux distribution with a different version as an OS. Mininet also possesses the capability to use Fedora Linux OS but with a few limitations in supporting numerous features. Moreover, it uses lightweight virtualization methods to offer efficient test environment in a single test machine, which in turn provides a developer or researcher a wide-ranging vision about real testbed network performance.In Mininet, controllers as well as switches can be installed and run either on user or kernel namespace to accelerate performance. Furthermore, every single host in Mininet represents an individual shell process that acts like an authentic real machine, which can be used to represent either a terminal edge in network or a server program such as HTTP and FTP servers. A host in Mininet unequivocally connects with the main Mininet module.Figure 2 shows an

example of a small network in Mininet. Network elements can interact with each other, and debugging is carried out using command line interface. This method of virtualization delivers preeminent sharing mechanism for resources of OS, rapid network booting, and easy to setup network elements.

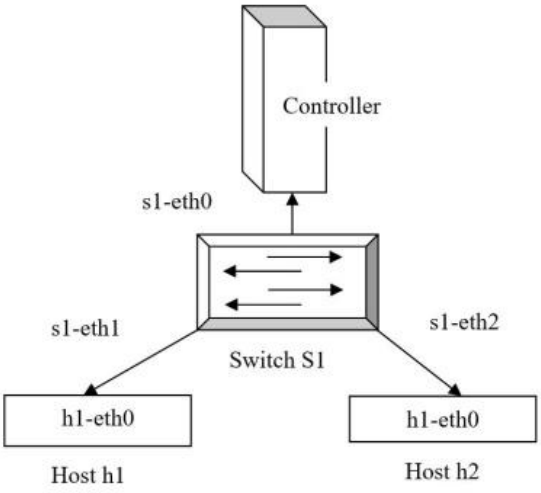


Fig. 2. Example of a small network in Mininet

Conversely, Mininet suffers from limitations like scalability for a large network that entirely relies on emulating machine specifications such as CPU type and frequency and system and RAM memory size [13]. The hardware and software resources that were used in this research were Intel Core™ i5-3320 MB CPU@2.60GHzx4 with RAM: 8GB, 128 GB SSD, 500 GB hard disk with Ubuntu 12.04 LTS-32-bit, and a kernel Linux 3.11.0-15-generic as OS.

B. Methodology

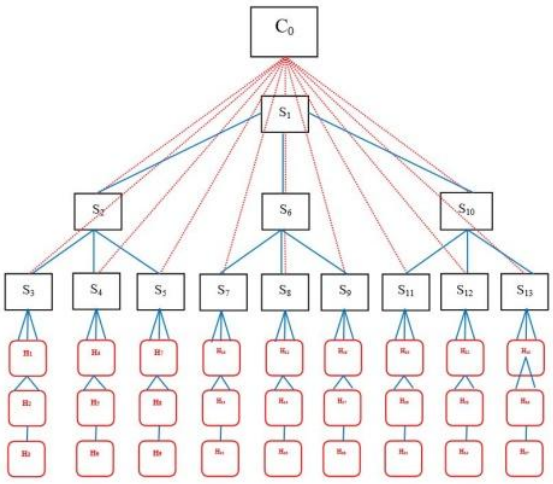


Fig. 3. Tree (3, 3) network topology

Tests in this research consist of the effect of load balancing scheduling algorithms with changing CPU time values. Mininet package has numerous Python script examples that are used to define Mininet capability, and a user can alter them to create contemporary functionality. To instigate an affluent method of designed test in Mininet, .py script was modified to implement tree topology (depth = 3, fan out = 3), as shown in Figure 3, where the network has 10 switches and 8 hosts in three cases. First, an OpenFlow network was built that used Open vSwitch, which supports v1.3 and is controlled by either OpenFlow remote controller or control and administration tool to augment practical flow entries. Second, an Open vSwitch was used in standalone mode (i.e., non-OpenFlow) to epitomize traditional network. Third, the network possesses OpenFlow as well as non-OpenFlow switches to exemplify a hybrid network. Note that end-to-end throughput is calculated for each network case (i.e., testing is performed between first host in network h1 and last host in network h27) by means of scheduling algorithms with changeable CPU time values (i.e., from 1% to 100%).

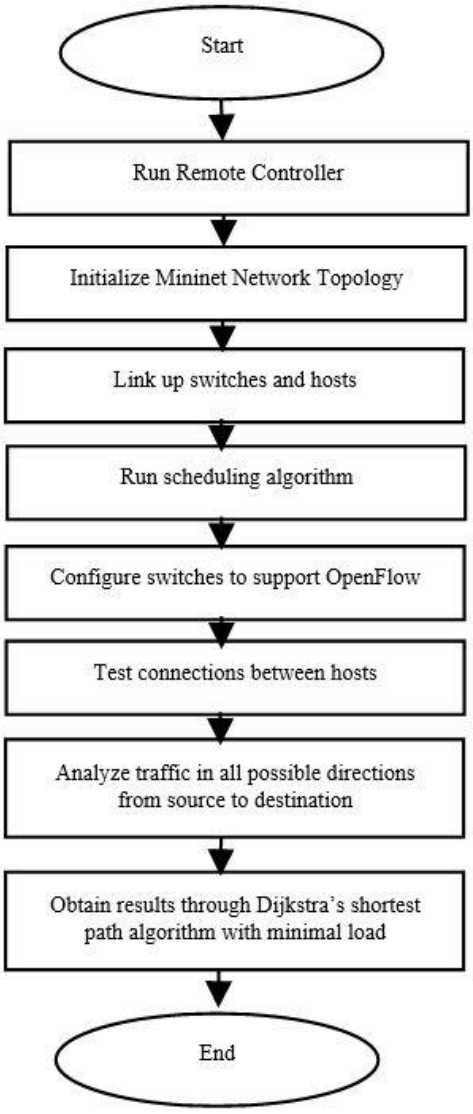


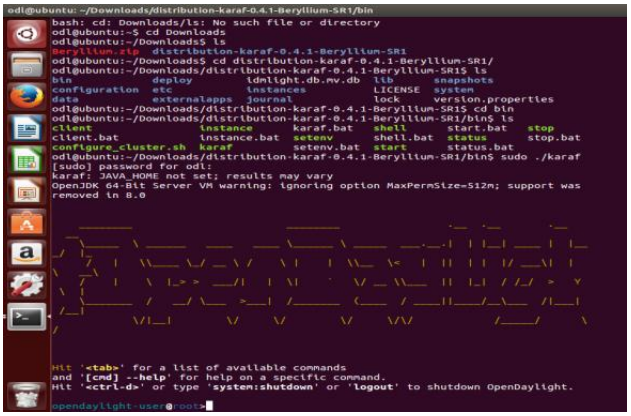
Fig. 4. Methodology of scheduling algorithm tests using remote controller.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

VI. RESULTS AND DISCUSSIOON

Step 1:Beginning OpenDaylight

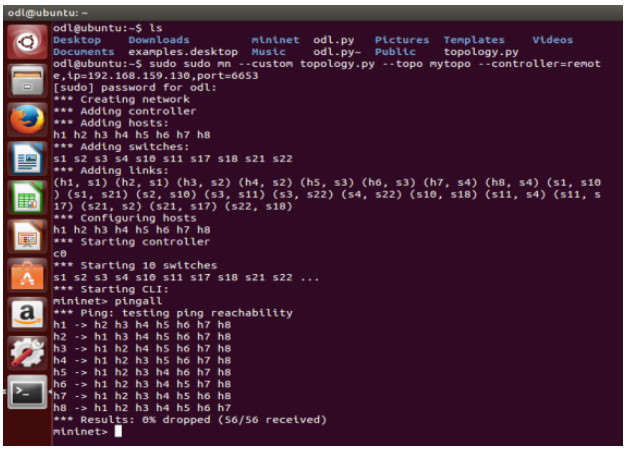
OpenDaylight is used as a controller running on Mininet OS.



Step 2:Create custom topologies

Custom topologies can be easily created through Mininet. For instance, creating custom topologies for 10 switches and 8 hosts require a few lines of coding in Python. Note that complex, flexible, and robust topologies can also be created. Moreover, such topologies can be configured based on several parameters that are to be passed, and topologies can be reused for several experiments.

Mininet connects to OpenDaylight Controller and establishes a three-level tree topology.



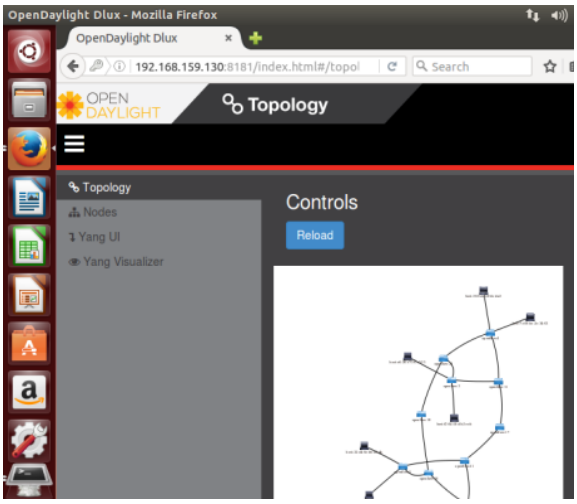
Step 3:Display topology via YANG data model

With OpenDaylight Controller and Mininet running, as illustrated in previous sections, log into web interface.

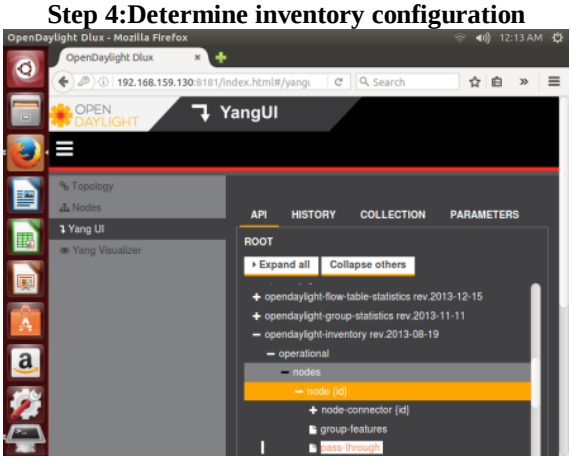
YANG is a modeling language written to support network configuration protocol (NETCONF)-based devices. However, in OpenDaylight, this model is used to describe the structure of data provided by controller components.

• **Config Generator:** It uses NETCONF (i.e., it provides a NETCONF endpoint). In addition, Config subsystem uses models to generate code. The scope of config generator is to make available dependency injection and configuration during the runtime of a server, which is to a certain extent similar to an outline in open service gateway initiative but externalizes XML from jar files. Consequently, a running OpenDaylight can be connected via NETCONF and push XML that reconfigures the server. For instance, mounting of a new Netcong/bgp/openflow node needs to be carried out, simply connect the node via NETCONF and push the change. This is opposite of how blueprint works, because in blueprint, XML for each bundle needs to be edited, repackaged, and deployed, and then the container needs to be started. YANG in config subsystem is specifically used to generate config java skeletons as well as type safe layer so that NETCONF client can perform partial validation.

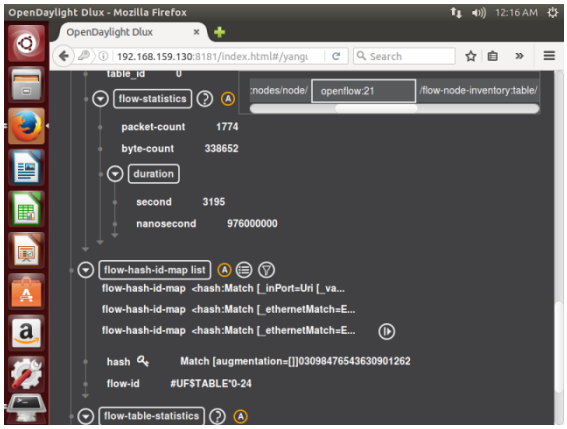
Config generator is used to describe the structure of data provided by controller components.



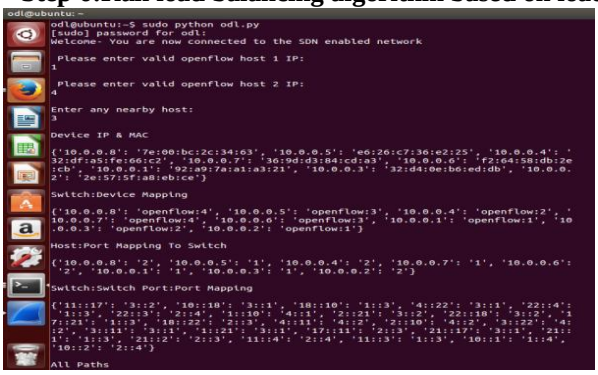
Step 4:Determine inventory configuration



Step 5:Push flow and test connection



Step 6:Run load balancing algorithm based on load



Sending the flow from host 1 to 4, IP addresses for host to switch, switch to switch, and controller to switch link connection can be obtained, as shown below.

Step 7:Obtain result by using optimal path based on load-based balancing algorithm

Two paths are available on host 4 (i.e., from switches 10 and 21), and then based on traffic, Dijkstra's shortest path algorithm can be applied and the final path with minimum cost can be selected.

VII. CONCLUSION

This study identified the following benefits that were obtained by deploying HPC applications on cloud: (i) automatic cost efficiency, (ii) straightforward use of HPC resources even without background knowledge in HPC cluster administration and maintenance, and (iii) resource scalability in cloud and the ability to scale up and down as per applications and user demand. However, HPCaaS needs to overcome the following challenges: (i) network latency and throughput, (ii) low performance due to cloud multitenancy, and (iii) low performance caused by virtualization overhead. The proposed system has scalable and dynamic architecture that makes use of SDN features. SDN-enabled task scheduling algorithm can consider VM cost and enable the scheduler to decide target VMs so that optimal performance can be achieved.

REFERENCES

1. Gupta, A., & Milojicic, D. (2011, October). Evaluation of HPC applications on cloud. In 2011 Sixth Open Cirrus Summit (pp. 22-26). IEEE.
2. McKeown, N. (2009). Software-defined networking. INFOCOM keynote talk, 17(2), 30-32.
3. Goecks, J., Nekrutenko, A., & Taylor, J. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome biology, 11(8), R86.
4. Gong, C., Liu, J., Zhang, Q., Chen, H., & Gong, Z. (2010, September). The characteristics of cloud computing. In 2010 39th International Conference on Parallel Processing Workshops (pp. 275-279). IEEE.
5. Expósito, R. R., Taboada, G. L., Ramos, S., Touriño, J., & Doallo, R. (2013). Performance analysis of HPC applications in the cloud. Future Generation Computer Systems, 29(1), 218-229.
6. Yang, C. T., Wang, H. Y., Ou, W. S., Liu, Y. T., & Hsu, C. H. (2012, December). On implementation of GPU virtualization using PCI pass-through. In 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings (pp. 711-716). IEEE.
   Nunes, B. A. A., Mendonca, M., Nguyen, X. N., Obraczka, K., & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. IEEE Communications Surveys & Tutorials, 16(3), 1617-1634.
7. Leite, A. F., Raiol, T., Tadonki, C., Walter, M. E. M., Eisenbeis, C., & de Melo, A. C. M. A. (2014, April). Excalibur: An autonomic cloud architecture for executing parallel applications. In Proceedings of the Fourth International Workshop on Cloud Data and Platforms (p. 2). ACM.
8. Gupta, A., Kalé, L. V., Milojicic, D. S., Faraboschi, P., Kaufmann, R., March, V., ...& Lee, B. S. (2012, June). Exploring the Performance and Mapping of HPC Applications to Platforms in the Cloud. In Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing (pp. 121-122). ACM.
9. Kim, H., & Feamster, N. (2013). Improving network management with software defined networking. IEEE Communications Magazine, 51(2), 114-119.
10. AbdelBaky, M., Parashar, M., Kim, H., Jordan, K.E., Sachdeva, V., Sexton, J., Jamjoom, H., Shae, Z.Y., Pencheva, G., Tavakoli, R. and Wheeler, M.F., 2012. Enabling high-performance computing as a service. Computer, 45(10), pp.72-80.
11. Taifi, M., Khreishah, A., & Shi, J. Y. (2013). Building a private HPC cloud for compute and data-intensive applications. International Journal on Cloud Computing, 3(2), 20.
12. Koldehofe, B., Dürr, F., Tariq, M. A., & Rothermel, K. (2012, December). The power of software-defined networking: line-rate content-based routing using OpenFlow. In Proceedings of the 7th Workshop on Middleware for Next Generation Internet Computing (p. 3). ACM.