HIGH PERFORMANCE COMPUTING AS A SERVICE IN THE CLOUD USING
SOFTWARE-DEFINED NETWORKING

Saba Jamaliannasrabadi

A Thesis

Submitted to the Graduate College of Bowling Green
State University in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE

August 2015

Committee:

Hassan Rajaei, Advisor
Robert Green
Jong Kwan Lee
ABSTRACT

Hassan Rajaei, Advisor

Benefits of Cloud Computing (CC) such as scalability, reliability, and resource pooling have attracted scientists to deploy their High Performance Computing (HPC) applications on the Cloud. Nevertheless, HPC applications can face serious challenges on the cloud that could undermine the gained benefit, if care is not taken. This thesis targets to address the shortcomings of the Cloud for the HPC applications through a platform called HPC as a Service (HPCaaS). Further, a novel scheme is introduced to improve the performance of HPC task scheduling on the Cloud using the emerging technology of Software-Defined Networking (SDN). The research introduces “ASETS: A SDN-Empowered Task Scheduling System” as an elastic platform for scheduling HPC tasks on the cloud. In addition, a novel algorithm called SETSA is developed as part of the ASETS architecture to manage the scheduling task of the HPCaaS platform. The platform monitors the network bandwidths to take advantage of the changes when submitting tasks to the virtual machines.

The experiments and benchmarking of HPC applications on the Cloud identified the virtualization overhead, cloud networking, and cloud multi-tenancy as the primary shortcomings of the cloud for HPC applications. A private Cloud Test Bed (CTB) was set up to evaluate the capabilities of ASETS and SETSA in addressing such problems. Subsequently, Amazon AWS public cloud was used to assess the scalability of the proposed systems. The obtained results of ASETS and SETSA on both private and public cloud indicate significant performance improvement of HPC applications can be achieved. Furthermore, the results suggest that proposed system is beneficial both to the cloud service providers and the users since ASETS performs better the degree of multi-tenancy increases. The thesis also proposes SETSAW
(SETSA Window) as an improved version of SETSA algorism. Unlike other proposed solutions for HPCaaS which have either optimized the cloud to make it more HPC-friendly, or required adjusting HPC applications to make them more cloud-friendly, ASETS tends to provide a platform for existing cloud systems to improve the performance of HPC applications.
Dedicated to my inspiring parents for their endless love and sacrifices
ACKNOWLEDGMENTS

All I have and will achieve no matter where I go is due to the unconditional support and endless love of my family. I would like to express my wholehearted thanks to my parents; Hassan & Marzieh Jamalian, my inspiring role-model brother; Nima and my kind beloved sister; Salma.

I would sincerely like to thank my supervisor Dr. Hassan Rajaei not only for his guidance, advice, and confidence in me, but also for his genuine support in my life. I cannot adequately express my thanks for his help and interest in seeing me succeed in all my endeavors.

I am very thankful to Dr. Robert Green and Dr. Jong Kwan Lee whose invaluable feedback, comments, and insights led to many improvements in the finished product.

And finally I would like to express my cordial thanks to my best friend, Rojin, for always being there for me, encouraging me in many moments of crisis, and helping me throughout the course of this research work.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER 1. INTRODUCTION</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1. Background</td>
<td>1</td>
</tr>
<tr>
<td>1.2. Foundations</td>
<td>2</td>
</tr>
<tr>
<td>1.3. Hypothesis</td>
<td>4</td>
</tr>
<tr>
<td>1.4. Research Goals and Objectives</td>
<td>4</td>
</tr>
<tr>
<td>1.5. Contributions</td>
<td>5</td>
</tr>
<tr>
<td>1.6. Thesis Organization</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER 2. HIGH PERFORMANCE COMPUTING AS A SERVICE</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1. Cloud Computing Overview</td>
<td>7</td>
</tr>
<tr>
<td>2.2. HPCaaS Platform</td>
<td>11</td>
</tr>
<tr>
<td>2.2.1. Benefits of HPCaaS</td>
<td>12</td>
</tr>
<tr>
<td>2.2.2. Challenges of the Cloud for HPCaaS</td>
<td>14</td>
</tr>
<tr>
<td>2.2.3. HPCaaS Solutions</td>
<td>18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER 3. SOFTWARE-DEFINED NETWORKING</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1. Why SDN?</td>
<td>22</td>
</tr>
<tr>
<td>3.2. SDN Architecture</td>
<td>25</td>
</tr>
<tr>
<td>3.3. OpenFlow</td>
<td>28</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER 4. RELATED WORKS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1. HPCaaS Review</td>
<td>29</td>
</tr>
<tr>
<td>4.1.1. Performance Analysis</td>
<td>29</td>
</tr>
</tbody>
</table>
4.1.2.  Performance Improvements ............................................................... 32
4.2.  SDN in the Cloud ..................................................................................... 35
4.3.  Performance Evaluation of SDN ............................................................. 38

CHAPTER 5. SDN EMPOWERED TASK SCHEDULING ........................................ 41
5.1  ASETS: A SDN Empowered Task Scheduling System ............................. 42
5.2  SETSA: SDN Empowered Task Scheduling Algorithm ............................ 44

CHAPTER 6. CLOUD TEST BED IMPLEMENTATION ........................................... 47
6.1  OpenStack .................................................................................................. 47
6.2  OpenDaylight ............................................................................................ 52

CHAPTER 7. CASE STUDIES AND PERFORMANCE EVALUATIONS .................. 54
7.1  Empirical Studies ....................................................................................... 54
7.2  Implementation Methods .......................................................................... 58
7.3  Experimental Results and Analysis ......................................................... 61
    7.3.1.  Measuring Overhead of SETSA .......................................................... 61
7.3.2.  ASETS on Private OpenStack Cloud .............................................. 63
7.3.3.  ASETS on Amazon Public Cloud .................................................... 64

CHAPTER 8. CONCLUSION ............................................................................. 66

CHAPTER 9. FUTURE WORKS ........................................................................ 68

REFERENCES .................................................................................................... 70
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Essential Characteristics of Cloud Computing</td>
<td>8</td>
</tr>
<tr>
<td>2.</td>
<td>Cloud Computing Service Models</td>
<td>10</td>
</tr>
<tr>
<td>3.</td>
<td>Speed-up for Matrix Multiplication benchmark running on Amazon EC2</td>
<td>15</td>
</tr>
<tr>
<td>4.</td>
<td>The efficiency achieved by running Matrix Multiplication benchmark on Amazon EC2 public cloud</td>
<td>16</td>
</tr>
<tr>
<td>5.</td>
<td>MPI Benchmark on 10G Amazon EC2 Virtual Network</td>
<td>17</td>
</tr>
<tr>
<td>6.</td>
<td>Traditional Network Routing</td>
<td>24</td>
</tr>
<tr>
<td>7.</td>
<td>SDN Controller in a Network</td>
<td>26</td>
</tr>
<tr>
<td>8.</td>
<td>SDN Logical Architecture (photo courtesy of ICCLAB)</td>
<td>27</td>
</tr>
<tr>
<td>9.</td>
<td>OpenFlow Switch Architecture</td>
<td>28</td>
</tr>
<tr>
<td>10.</td>
<td>An example of a problem with SDN controller</td>
<td>39</td>
</tr>
<tr>
<td>11.</td>
<td>ASETS Architecture</td>
<td>42</td>
</tr>
<tr>
<td>12.</td>
<td>Responsibility of Job Scheduler is to divide jobs into multiple tasks</td>
<td>43</td>
</tr>
<tr>
<td>13.</td>
<td>OpenStack Conceptual Services (Courtesy of openstack.org)</td>
<td>48</td>
</tr>
<tr>
<td>14.</td>
<td>OpenStack Dashboard Project Overview</td>
<td>51</td>
</tr>
<tr>
<td>15.</td>
<td>OpenDaylight Conceptual Architecture (Courtesy of.opendaylight.org)</td>
<td>53</td>
</tr>
<tr>
<td>16.</td>
<td>Results of Empirical Study Case Scenario I</td>
<td>55</td>
</tr>
<tr>
<td>17.</td>
<td>Results of Empirical Study Case Scenario II</td>
<td>56</td>
</tr>
<tr>
<td>18.</td>
<td>OpenStack and OpenDaylight Integration in a Private Cloud</td>
<td>59</td>
</tr>
<tr>
<td>19.</td>
<td>ASETS Conceptual Architecture</td>
<td>60</td>
</tr>
<tr>
<td>20.</td>
<td>Comparing the Performance of SETSA with FIFO in Multiple Experiments</td>
<td>62</td>
</tr>
<tr>
<td>21.</td>
<td>Performance of SETSA on Private OpenStack Cloud</td>
<td>63</td>
</tr>
</tbody>
</table>
22. The performance of SETSA on Amazon Public Cloud ............................................. 65
**LIST OF TABLES**

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. On-Premises HPC resources vs. HPCaaS</td>
<td>12</td>
</tr>
<tr>
<td>2. List of Tasks in the queue for empirical study case scenarios</td>
<td>54</td>
</tr>
</tbody>
</table>
CHAPTER 1. INTRODUCTION

1.1. Background

Cloud Computing according to NIST [1] is a shared pool of configurable resources offering services with five essential characteristics: on-demand self-services, broad network access, resource pooling, measured service, and rapid elasticity. A sixth characteristic that is often not considered essential but rather as a side-effect of resource pooling regards multi-tenancy. Clouds can be deployed as private within an organization, public for general users, or hybrid of the two. Cloud is service oriented architecture with three well-known services models: software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS) [2]. Recently “as-a-Service” has become a trend where a particular service platform is prefixed to the term to provide additional services to the user community. The main idea is to use the flexibility of the cloud and its benefits. Furthermore, clouds keep the software and applications up to dated, providing a wide variety of choices and various platforms.

High Performance Computing (HPC) is one of the application domains where numerous users desire move to the cloud. Nevertheless, HPC traditionally utilizes dedicated hardware with batch processing. HPC applications often run with direct physical access to hardware (i.e. multi-core machines, grid and supercomputers) that provides best achievable performance. As a result, the traditional HPS platforms are extremely fine-tuned to receive best performance of parallel processing. Moving HPC applications directly to the cloud can face performance penalty and thus extinguish the advantage gained from the cloud services [3], [4].

One of the technologies used in this thesis is a new innovation in network management called Software-Defined Networking (SDN) [5]. SDN is an emerging technology in networking that fits very well with the network virtualization of the cloud. SDN decouples the intelligence of
the network from the data plane and has it as a separate layer called the controller plane. The SDN controller rules the network policy and routing strategies by issuing commands and submitting statistical queries [6]. Such capabilities of the SDN have been used in this thesis to improve the performance of HPCaaS by better scheduling data-intensive HPC tasks on the cloud.

### 1.2. Foundations

Recent developments in Cloud Computing (CC) have attracted many sciences and industries to deploy their applications on the cloud. High Performance Computing (HPC) users and developers also have targeted CC as the base infrastructure for their applications. Accordingly, a new service called HPC as a Service (HPCaaS) appears to facilitate the execution of HPC applications on the cloud [7]. HPCaaS enables users to have on demand access to a scalable and reliable pool of high performance computing resources. However, HPC applications have unique requirements such as batch processing and dedicated hardware in order to achieve excellent performance. The issue has been addressed by either making the cloud systems more HPC-friendly [8] or optimizing a specific set of HPC applications to be more cloud-friendly [9]. In both cases, either the cloud or the HPC applications need to be altered in order for HPCaaS to have a competitive performance. Nevertheless an HPCaaS platform that targets the challenges of HPC applications on the cloud without modifying the cloud systems or the HPC applications is missing. This thesis aims to resolve the shortcomings of the cloud for executing HPC applications by providing an acceptable platform for HPCaaS.

One important challenge of executing HPC applications on the cloud is the high latency of the network caused by shared hardware in the multi-tenant environment of the cloud. Available bandwidths of the network links in the cloud rapidly change and lower the performance of parallel and distributed HPC applications. As a result, HPC applications on the
cloud can have an unpredictable performance depending on the number of tenants sharing the same infrastructure. This thesis proposes “ASETS: A SDN-Empowered Task Scheduling System” which is a novel scheme in HPC task scheduling on the cloud that benefits from a SDN controller to mitigate the problem of cloud networking for HPCaaS. Moreover, a task scheduling algorithm (“SETSA: SDN-Empowered Task Scheduling Algorithm”) that runs on top of ASETS is proposed to utilize available network bandwidths. By monitoring network links, SETSA schedules tasks on the most appropriate links based on the available bandwidth.

An intensive literature review confirms the novelty of the research and the design. In order to evaluate the performance and efficiency of the proposed systems, an implementation of ASETS and SETSA have been deployed on both a private Cloud Test Bed (CTB) and a public cloud. The CTB is powered by OpenStack [10] cloud operating system coupled with OpenDaylight [11] as the SDN networking module. The testbed is launched in the Distributed Systems laboratory of Computer Science Department on a cluster of 6 compute nodes. To scale up the performance and benchmarking experiments of ASETS and SETSA, Amazon AWS [12] as a market popular public cloud environment is utilized. Amazon limits access to the core functionalities of the cloud and hypervisor; therefore, to deploy the implementation of ASETS that utilizes a SDN network controller, a virtual OpenStack cloud is installed on Amazon EC2.

The result of the performance analysis shows a significant performance improvement of HPC applications on ASETS platform using the SETSA task scheduling algorithm. The benchmarking experiment indicates a low overhead of ASETS on the system and shows that SETSA performs better as the number of shared tenants increase. Consequently, SETSA enables cloud service providers to maximize benefits by sharing resources between more user tenants. In
addition, SETSA improves resource utilization and scalability of the HPC applications for HPC users.

1.3. Hypothesis

The hypothesis of this study is that Cloud Computing has great potentials to provide services for HPC applications. In addition, Software-Defined Networking can help to optimize the cloud network bandwidth, and thus benefit the task scheduling of HPCaaS by minimizing the turnaround time and maximize throughput.

1.4. Research Goals and Objectives

This research targets to provide an acceptable platform for HPCaaS by finding appropriate answers to the following research questions:

1. Pin point the primary challenges and shortcomings of executing HPC applications on a typical cloud.
2. Find the recent technological advancements and research achievements in the area of HPCaaS.
3. Discover methods to improve the performance of HPC applications on the cloud.
4. Set up a private Cloud Test Bed (CTB) for the cloud related research experiments.
5. Investigate limitations of the cloud networking for network intensive HPC applications.
6. Evaluate the performance of HPC applications on a traditional HPC cluster, a private cloud, and a public cloud.
1.5. Contributions

An intensive literature studies were conducted to facilitate finding the answers to the research questions of the thesis. Large amount of data were collected and analyzed, a cloud test-bed was set up, and case studies and performance analysis were carried out. As a result, this thesis secured to reach beyond its research goals and brought several novel approaches listed below:

1. Introduced ASETS (A SDN-Empowered Task Scheduling System) as a task scheduling scheme for HPC data-intensive tasks on the cloud
2. Introduced SETSA (SDN-Empowered Task Scheduling Algorithm) as a task scheduling algorithm to run on top of ASETS.
3. Introduced SETSAW (SETSA Window) as a parallel implementation of SETSA to improve performance and reduce overhead.
4. Deployed a private Cloud Test Bed (CTB) using OpenStack as the cloud operating system in the Distributed Systems laboratory.
5. Implemented ASETS and SETSA on a private OpenStack cloud using OpenDaylight to enable SDN for OpenStack networking as well as on Amazon AWS public cloud to evaluate the overhead, performance, and scalability of the proposed systems.
6. Conducted several benchmarking and performance evaluation experiments to compare the performance of HPC applications on a private OpenStack cloud, a public Amazon AWS cloud, and a traditional HPC cluster to identify the primary shortcomings of the current clouds for HPC applications.
7. Conducted a state-of-the-art literature review in the field of HPCaaS.
1.6. Thesis Organization

Chapter 2 is dedicated to the comprehensive literature review of the subject, chapter 3 explains HPCaaS in more details and chapter 4 describes the technology of SDN. Chapter 5 is dedicated to the details of ASETS and SETSA as well as the experiments conducted to identify the primary challenges of HPCaaS. Chapter 6 explains the implementation of the private cloud test-bed. Chapter 7 lists the case studies and chapter 8 is the results of the performance evaluation experiments of the system. Chapter 9 concludes the thesis and chapter 10 lists the potential future works of the research.
CHAPTER 2. HIGH PERFORMANCE COMPUTING AS A SERVICE

Cloud offers computing services such as processing power, storage and network bandwidth to end users. This section describes how HPC resources can be served by the cloud and then discusses the benefits and challenges of it. However, before jumping into HPC services provided by the cloud, first we will have a brief overview on the definition of the cloud, its service models and different deployment types to have a more accurate understanding of cloud computing.

2.1. Cloud Computing Overview

Cloud Computing is a buzzword referring to offer on-demand access to a pool of configurable computing resources. The computing resources may be storage, services, network, applications, etc [13]. There are five essential characteristics that shape Cloud Computing:

1. On-demand self-service: Services of the cloud are ready for the user whenever the user demands, without having to have human interference.

2. Broad network access: Users of the cloud services are able to have access to the services over a broad network and by using their workstations, mobile or tablets, laptops, etc

3. Resource Pooling: Cloud services are pooled to serve several users in a multi-tenant environment. Resources are dynamically assigned and reassigned and there is only a high level of abstraction for the location of the resources.

4. Rapid Elasticity: Resources on the cloud appears to be unlimited for the user. According to the user demand or even automatically the resources can scale up or down.
5. Measured Service: There is a metering system in the cloud that monitors, controls and reports service usages for both users and providers.

![Diagram of Essential Characteristics of Cloud Computing](photo courtesy of SPC International)

Each of the above characteristics is essential to shape a Cloud Computing model. The services on the cloud can be in form of the following three major models:

1. Software as a Service (SaaS): This service model (aka “on-demand software”) refers to applications that are hosted by the cloud. Users access the software through a broad network (e.g. the internet) and using web-browser or a light program interface. Service usage is measured by the cloud and the software is deployed and maintained by the cloud provider. SaaS often address application end users. As a very good example of this service model we can mention Salesforce which provides on-demand Customer Relationship Management (CRM) solution on the cloud.
2. Platform as a Service (PaaS): A subset of the combination of programming languages, software deployment tools, libraries, services, etc. for application developers to develop their own or install acquired software on the cloud. This service provided by the cloud enables developers not to worry about the underlying infrastructure (i.e. servers, network, storage, operating system, etc.) and instead focus on the deployed application and configuration details. Google plays an important role in this area by its PaaS solution, Google App Engine (GAE) [14]. GAE provides the necessary to manage, debug and deploy user applications on Google Cloud platform.

3. Infrastructure as a Service (IaaS): Capabilities provided by the cloud in this service model is the fundamental computing resources such as processing, storage and network so that the consumers are able to configure them according to their demand to deploy any arbitrary software. The consumer does not have control over the cloud underlying infrastructure but has control over operating system, storage and deployed applications. Amazon Web Services (AWS) [12] with its Elastic Compute Cloud (EC2) is the current leader in IaaS technology. It provides a Xen-based virtualized infrastructure with world-wide secure data center coverage. Another good example of IaaS is OpenStack [10] which is an open source cloud computing platform backed by several companies and contributors. This thesis is using OpenStack as the private cloud platform to evaluate the performance of HPCaaS.
Figure 2 compares the 3 major Cloud Computing service models. As we go from IaaS to SaaS, more number of resources and capabilities are managed by the cloud vendor than the ones consumers have control on.

To sum up the overview of Cloud Computing, we need to mention different deployment methods of Cloud Computing:

1. Private Cloud: It is a cloud platform for an organization that can be managed either within the corporate network or externally by a third party. This model may not bring the cost efficiency of the cloud as companies still need to buy commodity hardware to maintain and
administrate. However, many organizations have serious data security and data privacy concerns. The primary reason for most of the companies that choose to have a private cloud is that they are not willing to give the control of their data and information to a third party. Moreover, several countries enforce by law that critical data must be managed and stored within the country borders. In such cases, private cloud is the best solution.

2. Public Cloud: It refers to large cloud platforms provided by vendors for global public use. This model is best to reduce capital expenditure and operational cost of a corporate. The service is often provided by the vendor on a pay-per-use basis.

3. Hybrid Cloud: Having critical information and applications in the private cloud within the corporate and having shared applications and non-critical data on a public cloud helps businesses to take advantage of a secure private cloud as well as cost benefits of a public cloud. This approach is called a Hybrid Cloud.

4. Community Cloud: This model can be considered as a private cloud shared between several organizations with the same policy. Data will still be private and secure within the community but the organizations can benefit from sharing the cost between each other.

2.2. HPCaaS Platform

Cloud intends to provide utility computing by providing several resources as a service. Moving HPC applications to the cloud and utilizing cloud infrastructures for HPC programs is called HPCaaS. Primary motivations for moving HPC tasks to the cloud are in line with the typical benefits of Cloud Computing such as resource utilization, cost efficiency, flexibility, etc. Table 1 summarizes the comparison of the characteristics of an average on-premises HPC resource such as clusters or supercomputers with HPC provided by a public cloud (HPCaaS).
## Table 1. On-Premises HPC resources vs. HPCaaS

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>On-premises HPC resource</th>
<th>HPCaaS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Flexibility</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Reliability</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Cost</td>
<td>CAPEX</td>
<td>OPEX</td>
</tr>
<tr>
<td>Setup Time</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Maintenance &amp; Administration</td>
<td>expensive &amp; complex</td>
<td>N/A</td>
</tr>
<tr>
<td>Resource Utilization</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>

### 2.2.1. Benefits of HPCaaS

Besides general benefits of cloud computing, cloud is able to provide a number of unique advantages for HPC community that will be discussed in this section.

1. Ease of Maintenance and Administration: The majority of HPC users are biologists, physicists, and other scientists of academia not coming from a Computer Science background. They prefer not to get involved in the hassles of running and maintaining an HPC cluster or supercomputer. As of today most of the HPC users would need a server administrator to maintain the “on-premises” hardware and software resources. However, a public cloud provider preserves the infrastructure up-dated for running the applications keeping users away from the troubles of maintaining the resources [15].
Galaxy [16] is a very good example of a cloud based tool that provides a layer of transparency for computational research in the life sciences. The interactive web based tool makes it easy for biologists to use predefined software libraries and tools for their data intensive biomedical research.

One of the very popular life science algorithm commonly used in Bioinformatics is BLAST. Wei Lu et al. [17] utilized Microsoft Azure cloud infrastructure to implement and evaluate the performance of BLAST on the cloud. They proposed a cloud based architecture for BLAST that can be generalized to any cloud based infrastructure as well.

2. Efficiency in Resource Utilization: Clouds in general provide a dynamic and scalable infrastructure for business and web applications. In other words, according to the user and application demand, resources on the cloud can dynamically scale up and down. When it comes to HPC users in an academic setting, the variety of such spikes in the demand will increase. According to the timing of the scientific experiments, HPC users may not need the resources for a relatively long period of time or they may have a huge sudden demand for the resources time to time. As a result, utilizing cloud based resources and in particular benefiting from the pay-per-use model, will turn the capital expenses to operational expenses.

Cycle Computing is another example of transparent HPC on the cloud. They aim to provide utility supercomputing transparently on top of Amazon Infrastructure with an easy to use web based console. It makes it possible for scientists and HPC users to rent HPC resources worth of millions of dollars for a price of hundreds of dollars per hour [3].

3. Performance to Cost Ratio: As of today, a cloud based virtual cluster will provide a less performance in comparison with the physical cluster with the same configuration due to
several reasons discussed in section 5. However, it is a different story when it comes to the ratio of performance to cost [18]. The pay-as-you-go model and the scalability of the resources on the cloud can reduce the cost for running those of HPC applications with low communication overhead and I/O [15].

2.2.2. Challenges of the Cloud for HPCaaS

Several researches and studies compared the performance of HPC applications on the cloud with on-premises infrastructure and reached to this conclusion that clouds of today cannot compete with supercomputers [19], [20]. This section describes the primary reasons.

1. Multi-tenancy: As described in section 1, multi-tenancy is one of the characteristics of the cloud. It is also one of the profit making features of the cloud for cloud providers. It enables cloud providers to share resources between multiple tenants. Degree of multi-tenancy refers to the number of tenants sharing a same resource on the cloud. By increasing the degree of multi-tenancy, cloud providers are able to overprovision the resources to users. Overprovisioning allows cloud providers to maximize benefits, though with the risk of reducing QoS [21].

Nevertheless, multi-tenancy is in direct contrast with what HPC needs. HPC applications demand direct access to dedicated hardware using some sort of batch scheduling while shared resources on the cloud complicates the performance of HPC applications [22]. Shared resources make the simultaneous applications compete over resources resulting in lower performance of the applications as well as the network bandwidths.
Figure 3. Speed-up for Matrix Multiplication benchmark running on Amazon EC2 instance

We conducted an experiment by running a Matrix Multiplication benchmark on a virtual instance of Amazon EC2 public cloud. We repeated the experiment for 15 times and Figure 3 shows the result for the achieved speed-up. The focus of this experiment is not on speedup as Amazon AWS does not guarantee same physical configuration of processors in each virtual machine and each experiment. However, the experiment is focused on the performance predictability of HPC applications in Amazon public cloud. The error bars represent the standard deviation of the results and indicate that by increasing the number of cores, the diversity of values we get in multiple experiments, increases. Figure 4 is the efficiency achieved for the experiments and the error bars are again the standard deviations. This experiment shows how performance of HPC applications on the cloud is not predictable due to the shared resource and multi-tenant environment of the cloud. Moreover, the cloud lowers the efficiency.
There is a relatively huge gap between the average and best performance of running Matrix Multiplication benchmark on Amazon EC2. This gap is due to the fact that several tenants are using a shared resource and the performance of the application depends on the number of simultaneous running applications.

Other experiments such as [3] provide evidences that the multi-tenancy of the network is the major bottleneck and has the greatest influence in degrading the performance of HPC applications in the cloud.

2. Virtualization Overhead: Virtualization plays a key role in the cloud helping the cloud to have rapid elasticity, resource pooling, and flexibility. However, virtualization and in particular the hypervisor adds unwanted overhead by adding a software layer and preventing
applications to have direct access to the hardware resources. This virtualization overhead is not the same for all types of hardware. For example, because of the hardware support, virtualization overhead for processors is significantly less than the overhead of network virtualization. For some hardware types such as GPUs, it is often more efficient to pass through GPUs than to have virtual GPUs [23].

![ MPI Benchmark on 10G Amazon EC2 Virtual Network ](image)

**Figure 5. MPI Benchmark on 10G Amazon EC2 Virtual Network**

3. Network Bandwidth and Latency: Network interconnects and I/O resources in the cloud are shared between several tenants. Therefore, the bandwidth and latency of the network may not be predictable. The bandwidth in most cases is much less than what is expected. We conducted an experiment by running Point-to-Point MPI benchmarks [24] on a 10Gig Amazon EC2 interconnect. Figure 5 shows the achieved results. This experiment indicates that the multi-tenant environment of the cloud lowers the bandwidth. Moreover, the latency of the network on
the cloud is not stable. Therefore, we will see performance degradation for HPC applications and in particular data intensive ones.

2.2.3. HPCaaS Solutions

Several studies have tried to improve the performance of HPC applications on the cloud. This section categorizes these attempts into three groups based on the challenge they address: 1) Cloud-aware HPC applications, 2) Techniques to reduce virtualization overhead, 3) Network bandwidth and latency optimization.

1. Cloud-aware HPC Applications: We showed that the low performance of the networking in the cloud is the major bottleneck for HPC applications. Therefore, one technique for optimizing HPC applications for the cloud is to make them use the network more efficiently. Abhishek Gupta in his study [25] followed this approach successfully by tuning the computational granularity of the problem to get a higher performance.

Another area where we can alter our HPC applications to be adapted to the cloud is with new ideas of task scheduling. Unlike traditional homogenous clusters, cloud often provides highly heterogeneous resources for HPC applications. Another study has considered the heterogeneity property of the cloud when scheduling HPC applications [25]. Furthermore, traditionally, HPC users needed to specify the exact amount of resources (e.g. number of processors) before submitting an HPC job. Huang et al. [26] proposed a moldable job scheduling for HPC jobs on the cloud that relieves users’ burden of specifying number of processors besides achieving higher performance. Finally, Somasundaram et al. have proposed a framework for scheduling HPC applications in the cloud [27]. This framework benefits from Particle Swarm Optimization (PSO)-based Resource Allocation mechanism to minimize the job completion time.
Studies such as [28] argue that a hybrid environment of cloud with traditional HPC clusters will provide the best performance. They propose an HPC cluster that can intelligently scale up and use a public cloud infrastructure when needed. Our primitive studies also confirm that if we benefit from the advantages of traditional physical clusters alongside the flexibility and cost efficiency of the cloud, we may get a high performance.

Belgacem et al. in a more recent study [29] connected Amazon EC2 cloud resources located in USA to university clusters in Switzerland and ran a MPI based application. The result shows better performance for the application if a proper load balancing strategy is enforced.

2. Reducing Virtualization Overhead: Keller et al. proposed a new architecture [30] for Cloud Computing with no virtualization. This architecture removes the hypervisor and the virtualization layer and instead uses the hardware extensions to support the benefits provided by virtualization for the cloud. In this architecture only one VM runs on a core, memory is partitioned by hardware and all the devices are controlled directly by the VMs without any intervention of a hypervisor.

Palit et al. [31] also utilized hardware assistance techniques to reduce the overhead of virtualization. They argue that by incorporating the virtualization techniques into hardware chips, context switching will be accelerated, memory address translation will speed up and direct I/O access would become possible. In another study Renato Santos et al. [32] reduced the execution overhead of NICs to achieve a better performance for I/O virtualization near to direct I/O.

3. Network Bandwidth and Latency Utilization: Sonnek et al. [33] proposed an affinity-aware migration technique for allocation of virtual machines on available physical resources in such a way that the network bandwidths are efficiently utilized. In their experiment by running
the MPI benchmark, this technique could get up to 42% improvement in application’s runtime while reducing the network communication cost by 82%. Taifi et al. [34] also proposed architecture to build HPC clusters on top of a private cloud in which high throughput connections using Infiniband switches are used between the compute VMs whereas an Ethernet is connecting the compute nodes to the Cloud controller. The high bandwidth interconnects play an important role in improving the performance of the network which is very critical for HPC applications.

Link aggregation and altering the network topology according to the application demand may also reduce the network latency. Watanabe et al. in [35] improved the overall HPC performance by aggregating 4 to 6 links in a torus network topology.

4. Hybrid Cloud: One of the motivations of utilizing Cloud Computing for HPC applications is the scalability provided by on-demand resources of the cloud. HPC users traditionally run their applications on either their proprietary hardware or on Grid. With the advent of cloud computing, an architecture to combine all available HPC paradigms seems very beneficial. Mateescu et al. [36] introduced Elastic Cluster which basically provides a scalable hybrid infrastructure for execution of HPC workloads in a predictable manner. Predictable task execution means that the resources will scale from proprietary hardware to grid or a public cloud according to the granularity and characteristics of the HPC tasks and workflows. In a more recent study as well, Mantripragada et al. [37] are working on a method to run HPC applications on both an HPC traditional resource and a public cloud by partitioning the task and data of the application.
Our proposed solution to running HPC tasks on the cloud also benefits from being scalable. The scalability is in two levels; first is the ability to scale up and down inside the public cloud and second would be the capability of the system to scale up from a private cloud to the public cloud. In the first scenario, according to the number of tasks in the queue, virtual machines may be launched or terminated. This will make the cost more efficient as the user is not paying for underutilized resources. In the second scenario as well, if the system is running on a private cloud, virtual machines can be launched on a public cloud to be accessible remotely.
CHAPTER 3. SOFTWARE-DEFINED NETWORKING

Software Defined Networking (SDN) is a new networking paradigm and architecture recently come on the scene in which the network control and physical forwarding are separated apart. It is directly programmable, agile and manageable by a central software, open standard and vendor neutral network solution [38]. This chapter is dedicated to introduce this emerging technology by first give a brief history about the limitation of the traditional networking that led us to SDN. Then section 4.2 describes the architecture of SDN and finally section 4.3 gives an overview about the most popular SDN protocol; OpenFlow.

3.1. Why SDN?

Server virtualization has become increasingly popular recently. In spite of all the positive effects of server virtualization in data centers, virtual servers create problem with traditional network architectures. There are three big problems associated with server virtualization. First is the problem with configuration of virtual LANs. The network administrator has to make sure that the VLAN used by virtual machine is assigned to the same switch port that the host physical machine is plugged. Second problem with server virtualization is assigning QoS and enforcing network policies, such as access control lists (ACLs). This is traditionally done in network devices which we miss in virtual servers. Furthermore, in virtual servers, the traffic flows change in location and intensity over time. To address this demand, we need a flexible approach to manage network resources. Finally, the third big problem with server virtualization is the fact that the virtualized servers need a virtualized network that often comes with an unwanted overhead in implementation.
In network virtualization also, we sometimes need to insert services such as firewalls, load-balancers or routers between the various parts of the virtual network. This is referred to as service chaining [39].

With emerge of large enterprise data centers, network traffic patterns have dramatically changed. Communication is no longer limited to a form of a client-server application as today’s applications need to access different servers. “Big Data” requires high bandwidth and direct connection between thousands of servers. Operators at such datacenters constantly demand for additional network capacity to maintain any-to-any connectivity between the servers for transferring tremendous amounts of data.

Traditional computer networks maintenance and management is a challenging task for the network administrators as they have to deal with low level complex vendor-specific network devices to implement high level network policies [40].

Another area where SDN offers good solutions is to respond to rapidly changing resource and security requirements of mobile devices [41]. Respond to changing requirements in existing networking infrastructure can be very time consuming if the enterprise network is large. Current network hardware is horrible at maintaining connections when a user is moving. SDN helps the service providers to deal with such mobility problems.

In today’s networking, protecting data is a big issue. One very important concern in this regard is the ability to prevent the data cross the borders of nations. Several approaches including the application of SDN have addressed this problem (e.g., [42]).

Communication policies in traditional networks are defined before the network runs. However, with the use of SDN, one can change the entire services and policies of the network to
the current need in a matter of pushing a button. This enables the network administrator to shape the network in real-time. SDN helps to define the functionality of the network after it has been deployed [43].

Figure 6. Traditional Network Routing

Another issue with the traditional networking is the routing performance. Figure 6 shows a portion of a traditional core network. Imagine router A needs to send a packet to router E. First step for router A would be to send the packet to router B. Now when the packet is arrived at B, there are four different routing options in front of B. Router B is equipped with a routing algorithm and can be considered a smart network device. Based on the possible previous experience gained by the routing algorithm, router B knows that there is no path existed to router E from router F. Also, router B based on his knowledge will decide the most cost effective path to send the packet in order to get it to router E. Most probably, router B will choose router G for the next hop of the packet as its link seems to have the least cost in comparison to routers C and D. In this example router B has made a good choice, yet, it is not always the case. For instance, if
the path from router G to router E was 20 instead of the current 2, then the decision by B to send the packet to router G would result in the least performance for routing.

This example clearly shows, although routers in the current traditional networking are equipped with a routing algorithm and can be considered somewhat smart, the information they have from the network is very limited, therefore they may not guarantee the best performance of routing in the network. The limitations with the traditional method of network management for the routers are:

1. Routers only have the bandwidth and link costs of their directly connected links. They do not have a general understanding of the whole network and all the links.

2. Because of the inherent characteristics of the routing algorithms, the information routers have about the network is based on the history of the network and the results of the previous traffic. In other words, data of the routers about the network is always one step behind the decision time.

3. Each router has its own algorithm and intelligence. As a result the overall management of the network is decentralized and the exact behavior of the network is not predictable.

4. Any changes made to the network policy or changes in the architecture of the network take time to apply in the whole network.

3.2. SDN Architecture

The basic idea of SDN is to separate the control plane from the data plane and provide a layer of abstraction of the whole network for the user [5]. To our previous example of the traditional networking, we add a controller to the network that acts like an oracle knowing all the
details of every part of the network. This controller rules the network and is somehow connected to all the components of the network. When a packet arrives at a router, the router asks the controller what to do. In other words, the intelligence is taken away from the networking components and given to the controller. Since routers in this model are no longer smart, they can be called network switches. Figure 7 shows our traditional network structure but with a network controller added.

Figure 7. SDN Controller in a Network

Figure 8 shows a logical view of the SDN architecture. In a bottom-up approach, the infrastructure level is dumb network devices. The network equipment in this layer is dumb in the sense that they have become so simplified that they do not process any forwarding or routing policy for the incoming packets because the policies are forced from the control plane. The control and data planes communicate with each other using an interface. A very common and
well known communication protocol between these two layers so far is the OpenFlow which will be discussed in details in the next section.

The control layer is the centralized intelligence of the SDN. All the policies and network strategies for the whole enterprise network are defined in this layer. Users as well as the business applications of the application layer have access to this layer by the predefined APIs.

Figure 8. SDN Logical Architecture (photo courtesy of ICCLAB)
3.3. OpenFlow

OpenFlow is the practical implementation of SDN. In order to be able to control all the network devices from the management system, we need communication protocol. OpenFlow is a protocol to address this need. The OpenFlow protocol describes message exchanges that take place between an OpenFlow controller and an OpenFlow switch. The protocol is implemented on top of SSL or TLS to provide a secure channel.

This protocol enables the controller to perform add, update and delete actions to the flow entries in the flow table. Most modern network Ethernet routers and switches contain vendor-specific flow tables. OpenFlow provides an open protocol to program the flowtable in different switches and routers [6].

The datapath of an OpenFlow Switch consists of a Flow Table, and an action associated with each flow entry. Figure 9 illustrates an idealized OpenFlow switch. Controller communicates with the OpenFlow switch through a secure connection to update the flow table in the hardware level. The flow table dictates the behavior of the network.

Figure 9. OpenFlow Switch Architecture
CHAPTER 4. RELATED WORKS

4.1. HPCaaS Review

While Chapter 3 gives a detailed introduction to Cloud Computing and High Performance Computing on the cloud in form of HPCaaS, this section is dedicated to introduce the current state-of-the-art researches and studies in this field.

Cloud Computing continues to grow rapidly by redefining the way services are provided to end users. Desirable characteristics of the cloud such as on-demand access, resource pooling and cost efficiency, have tempted industries and academia to adapt some sort of this technology into their businesses. The results are promising and have proven the benefits of Cloud Computing. Recently, HPC users have also joined the large community of cloud users by moving their applications to the cloud. However, standard clouds do not satisfy all the unique requirements of HPC applications such as batch processing, access to hardware, bypassing OS kernel and high performance execution. Numerous studies are being conducted to address these challenges. In our literature review, we have categorized the researches about HPC in the cloud in two major groups: first is the performance analysis and second is the performance improvement of HPC applications in the cloud.

4.1.1. Performance Analysis

Abhishek Gupta et al. [4] have done an evaluation of cloud architecture for HPC applications in terms of cost efficiency. Open Cirrus [44] and Eucalyptus [45] were used as the cloud test-beds and their performance and cost were compared with a physical traditional HPC-optimized cluster. Although Open Cirrus showed a better performance than Eucalyptus in larger
scales, both cloud environments could not beat the performance of the physical cluster when the number of cores and size of applications grow. However, the analysis of the tradeoff between cost and performance showed that Clouds are more cost-efficient for embarrassingly parallel applications that do not need high network communications.

Qiming et al. in another study [19] evaluated the performance of Amazon EC2 as a public cloud infrastructure for HPC applications. This study identifies “poor network capabilities” as the primary problem of the cloud for running HPC jobs. They suggest that a cloud with a better network will provide a much better performance. As a secondary limitation of the cloud, this research points out the virtualization overhead in the Cloud. Benchmarks used in this research were all HPC benchmarks and different instance types in Amazon EC2 were tested to reach to an accurate conclusion. In a similar study [46], real world HPC applications were tested on Amazon EC2 and the results showed poor performance of the public cloud for HPC. This research also identifies network interconnects as the primary factor to limit performance. Again, in another similar research [47] Amazon EC2 public cloud infrastructure were evaluated for HPC applications. This time, Amazon public cloud was compared to NASA proprietary supercomputer for NASA HPC benchmarks. The results confirmed the previous conclusions of the poor networking. Moreover, virtualization overhead was mentioned as the secondary drawback of Amazon public cloud for running HPC applications.

Leite et al. in [48] proposed a cloud architecture named Excalibur for running parallel applications. The architecture includes a distributed file system, a job scheduler and a workflow management beside other components. The framework minimizes data movement to reduce cost and execution time and adjusts workloads to minimize job makespan. Moreover, the architecture has the capability of dynamically scaling up by adding more instances in an autonomic way.
Results of the autonomic configuration of the architecture showed 73% reduction in execution time and 84% reduction in cost.

Amazon, more recently, to address HPC unique requirements in the cloud, introduced compute optimized instances (Eight Extra Large instance cc2.8xlarge) with high speed network (10 Gigabit Ethernet). A study in [20] analyzed the performance of HPC applications using these types of instances. The result indicates that the performance of the communication has the most important role in defining the performance of HPC applications on the cloud. The performance of communication includes the network fabric as well as the support in the virtualization layer by the cloud hypervisor. In spite of the fast interconnects provided by Amazon, without a proper I/O virtualization support, a desirable performance for HPC applications is not achievable.

Another study [49] about the performance evaluation of HPC on the cloud addresses the application turnaround time and cost. This research compares Amazon EC2 public cloud with HPC clusters at Lawrence Livermore National Laboratory. Results were promising for the cloud as the turnaround time for some applications (e.g. Sweep3D [50]) was less than the turnaround time on physical clusters. Nevertheless, the raw performance of HPC clusters was superior to the cloud as expected.

Several other studies such as [18], [51], [52] have evaluated the cloud infrastructure for HPC and reached to this conclusion that in order for the cloud to be able to compete with traditional clusters for HPC applications, performance improvements and in particular network and communication enhancement are needed.

Our survey about the performance evaluation of HPC applications on the cloud and comparing it with traditional physical HPC clusters helped us have an understanding of the
current limitations and challenges for HPCaaS. Two primary challenges for HPCaaS identified by most of the studies are the networking and virtualization overhead. As part of this thesis we will implement a private cloud test bed on a proprietary cluster using OpenStack. This private cloud will be used to run several benchmarks to evaluate its computational power, memory, network and I/O access. The results will then be compared with the same cluster without cloud virtualization. This experiment will show how much is the overhead of having a cloud infrastructure.

4.1.2. Performance Improvements

Amazon EC2 as one of the current most popular and largest public clouds offers high performance CPU with large capacity of memory compute optimized instances at a higher price. For embarrassingly parallel jobs that do not need intensive network communications, results are promising. Amazon itself once built a supercomputer on its own cloud that could get into the list of the world’s Top 500 supercomputers. Cycle Computing [53] also in another effort built a big virtual supercomputer on top of Amazon cloud infrastructure with 30,000 cores for the price of $1,279 per hour.

Moustafa AbdelBaky et al. in [7] introduce a prototype framework to transform a supercomputer into a cloud that supports accessibility of HPC resources through IaaS, PaaS and SaaS abstractions. It is discussed in this paper that efforts to provide HPC resources for scientific applications in forms of “HPC in the cloud”, “HPC plus clouds” and “HPC as a cloud” have not reached to a complete success due to underlying commodity hardware’s limited capabilities, lack of high-speed interconnects, the physical (sometimes even geographical) distance between
servers and virtualization overhead. The study identifies ease of use, elasticity and accessibility of the cloud as the primary benefits of HPCaaS.

Moussa Taifi et al. in [34] propose an example of an HPCaaS architecture for building HPC clusters on top of a private cloud. High throughput connections (Infiniband) are used between compute VMs and an Ethernet network is connecting the compute nodes to the Cloud controller. These high bandwidth interconnects play an important role in improving the performance of the network which is very critical for HPC applications.

This study identifies three major benefits for HPCaaS; flexibility of the resources, resource efficiency and cost reduction. However, it is discussed that to achieve these benefits, there are three important challenges that need to be addressed; virtualization overhead, administrative complexity and a programming model. In order to address the administrative complexity of HPCaaS they have designed and implemented a virtual cluster administration tool called HPCFY. To evaluate the private HPC cloud both data-intensive (Hadoop sort and wordcount) and compute-intensive (MPI/OpenMP) benchmarks were used in the experiments. The results show that the primary weakness of the HPC cloud is to gain performance and reliability at the same time as it scales up. However, the study sees bright future for cloud-based HPC computing.

Task scheduling problem in a cloud environment is NP-Complete because both cloud resources and user application requirements dynamically change rapidly. Cloud Resource Broker (CLOUDRB) [27] is a proposed framework that is used to schedule HPC applications on the cloud for scientific purposes. The framework uses a Particle Swarm Optimization (PSO)-based Resource Allocation method to address the job scheduling problem. The purpose of the
framework is to minimize job finishing time, cost and job rejection ratio. Job rejection ratio is the rate of jobs that cannot be accepted by the cloud due to the unavailability of resources. On the other hand, the framework aims to maximize number of jobs finished within a deadline. The framework performance is evaluated by Matlab simulation as well as implementation on a private cloud environment with real-world HPC applications and the results showed a clear performance improvement as the framework meets the objectives.

The increasing demand to HPC in Biology has resulted in many approaches such as Galaxy [16] to address cloud based services for computation intensive algorithms in this field. Galaxy is a scientific workflow engine for computational biology with a web-based interface that makes it easy for biologists to run their applications on the cloud. Researchers at University of Chicago have used Galaxy to deploy bio-informatics workflow across local and Amazon EC2 cloud [54].

A more recent study [55] developed a proof of the concept framework for deploying HPC applications as a service on the cloud. The platform enables scientists and in particular Biology and Medicine specialists who have no computer science background to run their scientific HPC case studies easily and with promising performance.

Our literature review in regard to HPCaaS identifies the following benefits provided by deploying HPC applications on the cloud:

1. Automatic cost efficiency

2. Ease of use even for users with to background knowledge in HPC cluster administration and maintenance
3. Scalability of resources in the cloud and the ability to scale up and down according to application and user demand

However, HPCaaS still needs to overcome its shortcomings and challenges such as:

1. Network latency and throughput: Experiments discussed in Chapter 2, showed the fluctuating performance of the network in the Cloud. HPC applications often require high network bandwidths for synchronization and data transfer. The high latency caused by the rapidly changing network bandwidths in the Cloud makes the performance of the HPC applications highly unpredictable.

2. Low performance due to multi-tenancy of the cloud: Resources in the Cloud are often shared between several tenants using a virtualization layer. This is in contrast with the traditional demand of the HPC applications to dedicated hardware.

3. Low performance caused by virtualization overhead: Virtualization plays a key role in enabling multi-tenancy in Cloud Computing. However, it comes with an unwanted overhead for the operating systems and running applications in the Virtual Machines.

4.2. SDN in the Cloud

With the popularity of server virtualization, traditional networking techniques could not satisfy the requirements of the virtual network between virtual machines. Virtual networks demand the capability of dynamically changing network properties in run time [56]. The technology of Software-defined Networking (SDN) by separating the control unit can provide such functionalities for the network. SDN is a new born and young technology and ongoing researches are still being conducted on it. However, in recent three years, lots of applications as
well as prototyping are (and still being) proposed. Chapter 4 is dedicated to introduce the architecture of SDN in details. Nevertheless, this section presents the recent research and developments related to the application of Software-defined networking in the Cloud.

Boris Koldehofe et al. in [56] utilize the power of SDN and OpenFlow to develop a content-based routing. Content-based routing routes the message based on the content of the message, rather than by an explicitly specified destination. Utilizing SDN in enabling content-based routing minimize the bandwidth consumption in cloud networking and enhance the overall performance.

Prior to emerge of Software Defined Networking, an architecture [57] was proposed to manage and control routing in network based on a single controller. This architecture called Routing Control Platform has centralized controller collect information about the external destination and internal topology of an Autonomous System (AS) and then assigns the BGP based routes to each router. Rothenberg et al. proposed a methodology [58] to implement a routing control platform by using OpenFlow/SDN.

SDN also helps network security. Akbar et al. in [59] proposes a way to improve network security by detecting traffic anomaly in SOHO networks. They argue that with the application of OpenFlow and SDN, not only would not there be any performance reduction, but also detecting malicious activities becomes significantly more accurate.

In a local area network, in order to avoid loops in Ethernet topologies, the Spanning Tree Protocol (STP) is used. The best performance of this algorithm is of course achieved when this tree is a Minimum Spanning Tree. Prete et al. presented GreenMST [60] which is a simple OpenFlow based controller that creates a loop-free topology according to application on-demand
specifications. This has made the STP protocol have a better performance as well as more energy efficient.

One very premier purpose of SDN as mentioned before to make network management easier and more flexible. This can be done by dynamically shape the network according to on-demand complex QoS in real-time. A very good example of this is a prototyping [40] done by Kim et al. to attempt to solve three important network management problems; being tolerate to frequent network changes on-demand, high level network management, and better control for network troubleshooting.

A very interesting application of SDN in today’s networking is that SDN enables administrators to define network devices and functionalities such as firewall, load balancer, VLANs, etc, using software. This may not be as efficient in term of performance and accuracy as physical networking devices as of today; however, it is a very cost-effective solution for small businesses. Christopher Monsanto, et al. in [61] propose that by installing packet processing rules on open-flow enabled switches, on may be able to provide network functionalities such as load balancing using software.

Recently, it is proposed that SDN can help HPC too [62]. Nowadays, HPC applications are moving toward more distributed systems in which networking is a very important concept. A very interesting application of SDN in HPC is the HPC Job Management System proposed by Watashiba et al. [63]. In this JMS, network architecture and traffic is dynamically changed according the on-demand requirements of the HPC application. This method utilizes the network bandwidth toward achieving a higher performance.
4.3. Performance Evaluation of SDN

A final protocol and universal architecture for SDN is not released and introduced yet. This makes it very important for researchers to evaluate the rapid prototypes proposed for both SDN and its protocol; OpenFlow.

Kuzniar et al. [64] proposed an environment to make it easier to conduct systematically testing of SDN and OpenFlow. The work is still in progress and so far they have come up with a preliminary prototype. OFTest is another framework for testing SDN products. It is a python based application with a set of benchmarks to be used in testing OpenFlow switches. This framework was used primarily in testing OpenFlow 1.1. The OFTest server connects to both data and control planes of the SDN architecture. The switch under test is in the middle and the OFTest server monitors all the commands and data passing between every single part of the network related to the switch.

Software-Defined Networking Research Group or in short; SDNRG [65], as a part of Internet Research Task Force (IRTF) is a group of researchers conducting experiments and studying to address the fundamental state of the art questions in the area of SDN. Since SDN is still a nascent idea, several open problems exist ranging from architectural and design to implementation. SDNRG intends to facilitate research toward solving such problems.

SOFT is another framework for testing OpenFlow switches [66]. The difference between the frameworks with the similar previous approaches is that this framework address the interoperability of the OpenFlow switches. The way SOFT works is by identifying the inputs which may cause different implementations of OpenFlow behave inconsistently. So far SOFT has recognized so many inconsistencies between the current OpenFlow switches.
In spite of the fact that centralizing the control plane in SDN has reduced the probability of programming bugs, the system is still distributed and therefore complex. NICE [67] is another framework for testing SDN/OpenFlow architectures to address this issue. Figure 10 shows an example of such errors. It shows a flow path of a packet from host A to B. Both switch 1 and 2 are OpenFlow enabled switches that follow the networking rules from the controller. Before the packet reaches switch 1, OpenFlow controller had started installing the networking rule to the switches. When the packet passes switch 1, the rule is already installed and the switch works fine. However, when the packet reaches the second switch, due to the networking delay, rules in switch 2 are not updated and installed yet. Hence, the packet at this point is discarded as there is no destination defined for it. NICE tries to address such problems and issues.

![Figure 10. An example of a problem with SDN controller [67]](image)

CBench [68] and OFlops [69] are two other well-known and popular frameworks for testing SDN architectures. Cbench emulates any number of OpenFlow switches to measure different performance aspects of the controller including the minimum and maximum controller response time, maximum throughput, and the throughput and latency of the controller with a bounded number of packets on the fly. OFlops (Open Flow operation per second) is a modular framework and as a matter of fact an OpenFlow controller to benchmark various aspects of an
OpenFlow switch. Both OFlops and CBench are open-source frameworks and publicly available under the license of GNU.

Another work in the area of benchmarking SDN/OpenFlow architectures is the flexible OpenFlow-Controller benchmark proposed by Jarschel et al. [70]. This benchmark focuses on the controller part of SDN architecture. The controller behaves like the operating system of the whole SDN architecture, therefore has a significant influence on the performance of the whole network. In this methodology, in order to evaluate the performance of OpenFlow controllers, virtual OpenFlow switches are defined that generate OpenFlow messages to the controller.
CHAPTER 5. SDN EMPOWERED TASK SCHEDULING

Our literature review as well as the benchmarking experiments showed that the three most important and primary challenges of the Cloud for HPCaaS in current cloud infrastructures are the followings:

1. Virtualization Overhead: Hypervisors act as a middleware between the virtual machine and the bare underlying infrastructure preventing the VMs to have direct access to hardware. This cannot be a serious problem for day to day business applications, however, when it comes to HPC applications and in particular hardware-optimized HPC applications, this added overhead becomes an issue. Efforts such as [31] hardware-assisted virtualization and pass-through for hypervisors [23] have shown improvements.

2. Multi-tenancy of the Cloud: HPC applications often need batch processing. Nevertheless, current cloud architecture most of the times provides a multi-tenant environment for users. This multi-tenancy affects the performance in terms of computation power and also networking. Compute optimized instances address the issue of simultaneous access to processors by providing dedicated CPU usage. However, the problem with rapid and unpredictable changes of network connections in the cloud remains unsolved.

3. Cloud Networking: Shared network resources on the cloud cause high latency and unpredictable available bandwidth. This will make HPC applications have an unstable performance on the cloud [71].

This thesis aims to address the networking issue for HPCaaS on the cloud by leveraging the technology of Software-defined Networking. An elastic task scheduling algorithm on a typical HPCaaS configuration with shared memory File System is proposed. The algorithm
benefits from an SDN controller that has an overview of the network and in particular all the link bandwidths.

Figure 11. ASETS Architecture

5.1 ASETS: A SDN Empowered Task Scheduling System

Multi-tenant network in the cloud results in unpredictable and random link bandwidths with high average latency. Although, it would be very difficult to control the rapid changes in the bandwidth unless we dedicate a link to only one tenant, it is easier to monitor these changes using APIs of SDN controller and take actions accordingly. SDN controller is capable of sending queries about network statistics such as available network bandwidth on the links to OpenFlow enabled switches in a software-defined network.
We propose a task scheduling system that benefits from the SDN technology to be aware of the current bandwidth of the system. The architecture utilizing a SDN controller is shown in Figure 11.

The Job Scheduler has the responsibility of dividing HPC jobs into several tasks and putting them in the queue for the task scheduler. Tasks containing metadata of the needed data from the shared file system are assigned to an appropriate worker VM by the task scheduler that is taking advantage from the SDN controller to be aware of all the links bandwidths. A task scheduling algorithm runs in the task scheduler and not in the job scheduler. Figure 12 shows a conceptual view of the responsibility of the job scheduler.

![Figure 12. Responsibility of Job Scheduler is to divide jobs into multiple tasks](image)

Virtual machines are either in running state or waiting state. Running state is when the VM is launched and it is either running a task or ready to get a task to run (idle mode). Waiting state is when the virtual machine is not launched yet. For each empty port of a switch, we can assume a virtual machine in a waiting state.
5.2 SETSA: SDN Empowered Task Scheduling Algorithm

SETSA runs in the Task Scheduler in our proposed architecture. The task scheduler is linked to the SDN controller that is always aware of the network properties. The definitions of the variables in the algorithm are as follows.

- $D_j =$ Size of data for task $j$ (Bytes)
- $A_i =$ Next available idle time for VM $i$ (wall clock)
- $L_i =$ Time to Launch VM$_i$ (duration time)
- $DT_{ij} =$ Transfer time for data of task $j$ to VM $i$ (duration time)
- $T_{ij} =$ Time to execute task $j$ on VM $i$ (duration time)
- $Time_j =$ Time that task $j$ is finished (duration time)
- $M =$ Number of tasks
- $N =$ Number of Virtual Machines

Using SDN APIs, it is possible for the task scheduler to calculate $DT_{ij}$:

$$\forall i = 1 \text{ to } N, \forall j = 1 \text{ to } M; \quad DT_{ij} = \frac{D_j}{\max(BW, SF \rightarrow VM)}$$

For the VMs that are not launched yet, the next available idle time ($A_i$) would be equal to the time needed for them to launch and become ready ($L_i$)

Having these equations in mind, the task scheduler is able to calculate the finishing time of task $j$ on all the VMs and choose the one with the minimum value:

$$\forall i = 1 \text{ to } N, \forall j = 1 \text{ to } M; \quad Time_j = \min_{i = 1 \text{ to } N} (A_i + DT_{ij} + T_{ij})$$

Having the formulas in mind, the algorithm can be summarized as below:
Comparing SETSA with traditional FIFO or Round-Robin task scheduling algorithms, the following benefits provided by SETSA can be recognized:

1. Elastic Task Scheduling in HPCaaS: The ability to dynamically launch and stop virtual machines to execute tasks based on the number of requested tasks, makes the architecture elastic. This elasticity will help resource efficiency and cost efficiency.

2. Improved Performance of HPCaaS with SETSA: SETSA monitors network links and bandwidths to apply them in the task scheduling process. When the task scheduler assigns tasks to VMs, it takes the link bandwidths into the account. This information helps SETSA to assign the task to a VM to which data can be transferred more efficiently.

3. Improved Utilization of the Network Bandwidth: SETSA attempts to use under-utilized links for data transfer.

---

**Algorithm 1 SETSA**

1. \textbf{while} \textit{true} \textbf{do}
2. \hspace{1em} \(j \leftarrow \text{pop a task from tasks queue}\)
3. \hspace{1em} \textbf{Let} \(N\) \textbf{be the number of VMs}
4. \hspace{1em} \textbf{for} \(i = 1\) \textbf{to} \(N\) \textbf{do}
5. \hspace{2em} \textbf{if} \(VM_i\) \textbf{is not launched} \textbf{then}
6. \hspace{3em} \(A_i \leftarrow L_i\)
7. \hspace{3em} \(BW_{ij} \leftarrow \text{Available Bandwidth}\) \hspace{1em} \(\triangleright \text{OpenFlow API is used here}\)
8. \hspace{3em} \(DT_{ij} \leftarrow D_j \div BW_{ij}\)
9. \hspace{3em} \(Min \leftarrow \infty\)
10. \hspace{3em} \(Time_j \leftarrow A_i + DT_{ij} + T_{ij}\)
11. \hspace{3em} \textbf{if} \(Time_j < Min\) \textbf{then}
12. \hspace{4em} \(Min \leftarrow Time_j\)
13. \hspace{4em} \(VM \leftarrow i\) \hspace{1em} \(\triangleright \text{Task } j \text{ will be assigned to VM}\)
14. \hspace{3em} \textbf{if} \(VM\) \textbf{is not launched} \textbf{then}
15. \hspace{4em} \text{Launch VM}

---

Comparing SETSA with traditional FIFO or Round-Robin task scheduling algorithms, the following benefits provided by SETSA can be recognized:

1. Elastic Task Scheduling in HPCaaS: The ability to dynamically launch and stop virtual machines to execute tasks based on the number of requested tasks, makes the architecture elastic. This elasticity will help resource efficiency and cost efficiency.

2. Improved Performance of HPCaaS with SETSA: SETSA monitors network links and bandwidths to apply them in the task scheduling process. When the task scheduler assigns tasks to VMs, it takes the link bandwidths into the account. This information helps SETSA to assign the task to a VM to which data can be transferred more efficiently.

3. Improved Utilization of the Network Bandwidth: SETSA attempts to use under-utilized links for data transfer.
4. Reducing turnaround time in HPCaaS: By better utilizing the network links, SETSA will increase the performance in terms of job finishing time and turnaround time in HPCaaS.

Furthermore, there are the following potentials in SETSA that can be addressed as expansions:

1. Measure overhead of the SDN controller and attempts to reduce this unwanted overhead

2. Adding “cost” to the model to maximize performance/cost ratio for HPCaaS

3. Expand the model to apply to distributed storage architecture

4. Expand the approach to the Job Scheduler
CHAPTER 6. CLOUD TEST BED IMPLEMENTATION

6.1. OpenStack

The phrase “Cloud Operating System” for data centers, finely describes OpenStack. It is a free and open-source software released under the terms of the Apache License. Initially, OpenStack was first introduced by a joined community from Rackspace Hosting and NASA. As of today, more than 200 companies have joined the project, including IBM, Cisco, Canonical, Dell, HP, AT&T, VMWare, AMD, Intel, Orcale and Yahoo.

In September 2012, the OpenStack Foundation as a non-profit corporate entity was established to manage the OpenStack project. The project’s mission according to the OpenStack Foundation is to produce the ubiquitous Open Source Cloud Computing platform that will meet the needs of public and private clouds regardless of size, by being simple to implement and massively scalable.

The OpenStack system includes several independent but deeply related open source projects that work together to shape an Infrastructure as a Service (IaaS) cloud system. Each of the projects can be installed and deployed separately. Further, they can be configured to work independently as standalone applications or to work together. The OpenStack Foundation offers several in-depth documentations and manuals to install and deploy different parts of the OpenStack. A good example of such installation guides could be [72] that offers a step by step Installation guide for every component of the OpenStack on Red Hat Enterprise Linux, CentOS and Fedora.
OpenStack provides three major services; Computing, Networking and Storing. The followings will discuss each of these services as well as other services provide by OpenStack in more details.

![OpenStack Conceptual Services](image)

**Figure 13. OpenStack Conceptual Services (Courtesy of openstack.org)**

OpenStack Compute: The most important component of this service is the Virtual Machines (VMs). Virtual machines are flexible, scalable and can be created on-demand. The two most important projects involved in this service are Nova and Glance.

Nova: Nova or OpenStack Compute is an open source python based project and the main part of the OpenStack as an IaaS system. Nova provides facilities to launch and manage the virtual machines by providing an abstraction layer on top of the hypervisors. As of today, a large number of different hypervisors is supported by OpenStack including KVM, Xen, LXC, VMware. However, they do not all support the same features. KVM is the default hypervisor of OpenStack.
Glance: Glance is also an open source python based project. This software offers services for discovering, registering, retrieving and storing virtual machine images. The images may be stored either as simple files in a file-system based storage or as objects in an object-storage system. OpenStack Swift storage service offers an object storage system that can be used by Glance to store the VM machines.

OpenStack Networking: It is a pluggable, scalable and API-driven system for managing networks and IP addresses. The system allows the users to create their own networks and assign static, floating, or dynamic IP addresses to VM instances.

Users and administrators have two options to make when it comes to networking in OpenStack. One is the "legacy" Nova-Network and the other is Neutron. Nova-network is a much simpler way to deploy network, but does not have the full “Software Defined Networking” features of Neutron. Software-defined networking (SDN) is an emerging technology in network organization and management that benefits cloud networking as well.

Neutron: Neutron is an OpenStack Python based project to provide "networking as a service" between interface devices such as virtual network interfaces managed by other Openstack services such as Nova.

OpenStack Storage

This component provides block and object storage to use by VM instances. The block storage system called Cinder allows the users to create block storage devices and dynamically attach and detach them from VM instances using the dashboard or API. In addition to block storage, OpenStack provides a scalable distributed object storage called Swift, which is also accessible through an API.
Cinder: Cinder provides an infrastructure for managing volumes in OpenStack. It was originally a Nova component called nova-volume, but has become an independent project later. It is based on Python and the source code is freely available.

Swift: Swift is a distributed object storage system designed to scale from a single machine to thousands of servers. Swift is optimized for multi-tenancy and high concurrency. Swift is ideal for backups, web and mobile content, and any other unstructured data that can grow without bound.

OpenStack Dashboard: The dashboard is a graphical web-based interface to provide access for the users and administrators to use the core services provided by OpenStack components. As a cloud administrator, the dashboard provides an overall view of the size and state of your cloud. You can create users and projects, assign users to projects and set limits on the resources for those projects. Also, for the users, the dashboard provides a self-service portal to provision their own resources within the limits set by administrators.

Horizon: Horizon is the canonical implementation of Openstack’s Dashboard. Similar to other OpenStack projects, it is based on Python and the source code is freely available. Figure 14 shows a screen shot of a project overview and resource usage in Horizon. Horizon provides a detailed overview of the available resources in the Cloud and how the virtual machines are utilizing them.

OpenStack Identity Service: It is a common authentication system for all the OpenStack components in the whole cloud operating system. It supports multiple forms of authentication including standard username and password credentials, token-based systems and AWS-style logins. Moreover, it can integrate with existing backend directory services like LDAP.
Keystone: Keystone is the current identity service used by OpenStack for authentication (authN) and high-level authorization (authZ). It currently supports token-based authN and user-service authorization. It has recently been re-architected to allow for expansion to support proxying external services and AuthN/AuthZ mechanisms such as OAuth, SAML and openID in future versions. The source code is freely available.

Besides the aforementioned core-services, OpenStack has several other ongoing projects as well. This section describes a selected number of these several projects. Furthermore, proposals for future services and projects are suggested. As of today, cloud interoperability seems to be the priority of research and development. Many believe that the hybrid cloud will become the dominant computing model and is positioning accordingly with OpenStack.
OpenStack Orchestration and Heat: The mission of the OpenStack Orchestration program is to create a human- and machine-accessible service for managing the entire lifecycle of infrastructure and applications within OpenStack clouds. As of today, the main project in the OpenStack Orchestration is Heat. Based on a predefined template in form of a text file, Heat launches multiple cloud applications and VMs for a specific predefined purpose. The project is still in progress and the first version of it is used in the latest release of OpenStack (i.e. Havana).

OpenStack Telemetry (Ceilometer): One of the essential characteristics of a cloud system as mentioned before, is the ability to measure service usage and provide pay-per-use services. In order to turn that into reality, Ceilometer is aimed to deliver a unique point of contact for billing systems to acquire all meters they need to establish customer billing, across all current and future OpenStack core components. The project is still in progress. Its primary targets are monitoring and metering, but the framework should be easily expandable to collect for other needs. The project source code which is again based on Python is freely available.

OpenStack Common Libraries (Oslo): To produce a set of python libraries containing code shared by OpenStack projects. The APIs provided by these libraries should be high quality, stable, consistent, documented and generally applicable.

6.2. OpenDaylight

OpenDaylight is an open platform for network programmability to enable SDN and NFV for networks at any size and scale. The community’s second release “Helium” comes with a new user interface and a much simpler and customizable installation process thanks to the use of the Apache Karaf container. OpenDaylight software is a combination of components including a fully pluggable controller, interfaces, protocol plug-ins and applications. With this common
platform both customers and vendors can innovate and collaborate in order to commercialize SDN- and NFV-based solutions.

Figure 15. OpenDaylight Conceptual Architecture (Courtesy of opendaylight.org)
CHAPTER 7. CASE STUDIES AND PERFORMANCE EVALUATIONS

7.1. Empirical Studies

In this section we empirically model the system in a small scale of 3 workers for demonstration purpose. Examples are made to illustrate and compare two case scenarios for the algorithm. For simplicity, we assume that the workers are homogenous; therefore the time needed to finish a particular task is equal for all the workers. In both case scenarios a group of 8 tasks with different data sizes are assumed.

Table 2 . List of Tasks in the queue for empirical study case scenarios

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Data Size (Mb)</th>
<th>Needed Time</th>
<th>Task ID</th>
<th>Data Size (Mb)</th>
<th>Needed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>7</td>
<td>5</td>
<td>1500</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>800</td>
<td>15</td>
<td>6</td>
<td>1800</td>
<td>35</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>2</td>
<td>7</td>
<td>500</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>1200</td>
<td>19</td>
<td>8</td>
<td>700</td>
<td>18</td>
</tr>
</tbody>
</table>

The two scenarios differ only in term of networking. In the first scenario, network bandwidth for the links connecting the shared file system to the workers, are assumed to be very close to each other. While in the second case, there is a significant difference between the links due to severe multi-tenancy of the system. We compare SETSA with FIFO and Round Robin for both scenarios in term of turnaround time.
Case Scenario I: In this example, network bandwidths for the links connecting the shared file system to virtual machines 1 to 3 are assumed to be 40Mbps, 50Mbps and 60Mbps. This is considered a small difference in the scale of a multi-tenant network of the cloud.

Figure 16. Results of Empirical Study Case Scenario I

Figure 16 illustrates the result for the 3 different task scheduling algorithms. SETSA shows same result with FIFO and a slight improvement over RoundRobin. When we scale up the system to hundreds of workers, this improvement becomes considerable. However, we expect SETSA to show a much more significant performance improvement when the degree of multi-tenancy in the cloud increases. Case scenario II investigates the effect of increasing multi-tenancy.
Case Scenario II: In this example, the network bandwidths for the links are assumed to be 5 Mbps, 100Mbps, and 20Mbps. These values are selected based on the fact that the cloud is fully utilized by many tenants. Results in Figure 17 are considerably promising.

![Graphs showing task execution time and network utilization](image)

Figure 17. Results of Empirical Study Case Scenario II

When the degree of multi-tenancy increases the SETSA algorithm improves the performance of HPCaaS in terms of task turnaround time and network utilization up to 75%. Our studies show that when increasing numbers of tenants are using a shared network and thus the cloud network becomes further unstable, SETSA will even play a more significant role in performance improvement.
Overhead of SETSA: SETSA in comparison with traditional scheduling techniques such as FIFO or Round Robin, adds more overhead as it needs repetitive API calls to SDN controller with $O(n)$ computation complexity for finding the best VM per task (where $n$ is the number of virtual machines). Furthermore, SETSA shows promising results when the network is unstable (i.e. has more burst traffic) and have an effect on the scheduling. In a network with stable bandwidths and latency, SETSA may not expect to show promising results. Nevertheless, if the degree of multi-tenancy in ASETS increases, the network will start to show instability in bandwidth and latency. Therefore, SETSA will show the performance improvements in the ASETS architecture.

This observation shows that there is always a threshold in the degree of multi-tenancy after which SETSA improves the performance of ASETS by increasing network throughput efficiency and decreasing task turnaround time. A recent study [21] indicates that cloud service providers tend to offer services to the users more than their actual capacity by using oversubscription in order to increase revenue. However, this oversubscription lowers the performance. Oversubscription in other words means increasing the degree of multi-tenancy. Therefore, SETSA can play a critical role in such systems in order to stabilize the performance while the cloud service provider may increase the revenue by oversubscription.

SETSAW: SETSA Window: In each generation of SETSA, only one task is being assigned to one virtual machine. We can improve the performance of SETSA by assigning multiple tasks to several virtual machines per iteration. Thus, we propose SETSAW (SETSA Window) in which a number of tasks (equal to window size) are assigned to the appropriate VMs at a same time. This improves the performance of SETSA and reduces its overhead. The details
of the SETSAW algorithm and its performance evaluations thereafter will be described in forthcoming papers. SETSW aims to improve SETSA in the following items.

1. Increasing efficiency by assigning multiple tasks to multiple virtual machines at a time.
2. Improving the performance with parallelization
3. Helping the scalability of the system by providing distributed scheduling mechanisms

Implementation of the SETSAW and performance evaluation of the system is an ongoing work and discussed as one of the future works of this thesis.

7.2. Implementation Methods

This section describes our methodology to implement ASETS on both Amazon public cloud and a private OpenStack cloud. Depending on the infrastructure and platforms, there are several technologies that can be used to implement ASETS. In order to show the proof of the concept, we deployed an OpenStack cloud integrated with OpenDaylight as the SDN enabler for the virtual network of the cloud.

Private OpenStack Cloud: We deployed a RedHat RDO on our Dell commodity cluster of 6 compute nodes to have a private OpenStack cloud. The OpenStack manages cloud networking using Neutron. In order to enable SDN on this cloud, we need to configure Neutron to work with OpenDaylight using Open vSwitch and Modular Layer 2 (ML2) plug-in. Figure 8 represents the conceptual overview of the integration of OpenStack with OpenDaylight to enable Software-Defined Networking for the private cloud. OpenDaylight connects with the networking module of OpenStack (Neutron) to provide SDN capabilities.
One of the primary challenges of evaluating the performance of ASETS and SETSA on a private cloud is to build a multi-tenant environment. SETSA improves the performance of HPCaaS if the cloud resources are utilized enough by simultaneous tenants. In order to emulate such an environment for ASETS we set up several virtual clusters of 3 to 4 small scale compute nodes each running a Matrix Multiplication benchmark. This challenge only needs to be addressed in a private cloud setting as the Amazon public cloud resources are already fully utilized by real working tenants. Another shortcoming of the private OpenStack cloud for our experiments was the small scale of the implementation. The private cloud was built on top of a cluster of 6 compute nodes and over a total of 64 physical cores. Although the hardware configuration was enough to prove the concept, a larger scale of experiments was needed to show the elasticity and feasibility of ASETS on a real-world public cloud environment. To address such a problem, we implemented ASETS on Amazon AWS cloud as well.

Public Amazon Cloud: A real-world multi-tenant and dynamic public cloud is desired to evaluate the performance of ASETS and SETSA more accurately. Nevertheless, public cloud providers such as Amazon will provide a limited access in the infrastructure and hardware layer
to the users, making it very difficult for us to deploy our SDN enabled cloud networking. Although, public cloud providers may utilize Software-Defined Networking capabilities for their networking infrastructures, such capabilities are blocked for public users for several reasons, primarily the security. To overcome this problem, we deployed a private virtual OpenStack cloud integrated with OpenDaylight on a virtual cluster on Amazon EC2. This will add another layer of virtualization to the system and therefore an unwanted overhead, yet makes it possible for us to utilize SDN capabilities for our own private cloud on top of a multi-tenant infrastructure to accurately evaluate performance of ASETS.

![Figure 19. ASETS Conceptual Architecture](image)

Figure 19 shows the conceptual architecture of our implementation of ASETS on Amazon public cloud. Amazon infrastructure provides a multi-tenant environment for us where we set up a private virtual OpenStack cloud with Software-Defined Networking enabled by OpenDaylight.

Amazon AWS provide cloud based services such as Amazon SQS (i.e. a queuing system) and powerful APIs besides typical virtual machines that make cloud-based developments a lot
easier. In our implementation of ASETS on Amazon AWS, we utilized Amazon SQS for the tasks queue. Moreover, the Amazon EC2 Java APIs enable us to dynamically launch and terminate virtual machines on the cloud. Utilizing this capability, in order to make ASETS scale up and down according to the number of incoming tasks, we developed a module that actively monitors the size of the tasks queue. If the number of tasks in the queue exceeds a threshold, ASETS automatically launches new virtual machines to scale up. On the other hand if a virtual machine remains idle for a specific period of time, ASETS will terminate the virtual machine to save cost.

7.3. Experimental Results and Analysis

We conducted the comprehensive performance analysis of ASETS and SETSA from three different perspectives; measuring the overhead of SETSA, performance evaluation of the system on a private cloud, and performance evaluation on Amazon public cloud. Our experiments indicate promising results for ASETS and its primary scheduling algorithm, SETSA. The proof of the concept implementation clearly indicates that ASETS is highly scalable and SETSA improves the performance of HPCaaS when the degree of multi-tenancy goes up.

7.3.1. Measuring Overhead of SETSA

Unlike regular scheduling algorithms like FIFO or RoundRobin, SETSA needs more calculations as it uses SDN APIs to monitor network bandwidths and make decisions accordingly. The extra calculations and process adds unwanted overhead that may influence the performance of the system. In order to measure the overhead of the system we compared the performance of ASETS when running SETSA with the time it is running a simple FIFO scheduling algorithm. The experiment was conducted on a private OpenStack cloud on a
commodity cluster of six compute nodes running six virtual machines with zero multi-tenancy and repeated for 10 times. Data sizes and task granularities were randomly chosen for each repeat of the experiment.

Figure 20 compares the performance of SETSA in 10 numbers of executions with FIFO in a private cloud with no multi-tenancy. When there is not any simultaneous applications running on the cloud, network bandwidths remain stable and SETSA schedules HPC tasks the same as FIFO. The experiment shows that the undesired overhead of SETSA running on ASETS in such a case is approximately 5%. Further studies and experiments will indicate that this overhead is reasonably low and worthwhile.

Figure 20. Comparing the Performance of SETSA with FIFO in Multiple Experiments
7.3.2. ASETS on Private OpenStack Cloud

Our empirical analysis of SETSA, previously, indicated that as the degree of multi-tenancy increases, SETSA performs better by mitigating the overhead of the multi-tenancy and improves the performance of HPCaaS. In order to evaluate SETSA in action, we artificially created a multi-tenant environment on our private OpenStack cloud by launching simultaneous virtual clusters. Each virtual cluster has 3 virtual machines and runs a Matrix Multiplication algorithm. Number of the virtual clusters running on our cloud indicates the degree of multi-tenancy.

![Figure 21. Performance of SETSA on Private OpenStack Cloud Based on the Degree of Multi-tenancy](image)

Figure 21 confirms our empirical analysis. When the degree of multi-tenancy is low, SETSA performs almost the same as FIFO. However, as the number of simultaneous applications running on the cloud goes up, SETSA tends to mitigate the fluctuating available
network bandwidths of the cloud and therefore increase the performance of HPCaaS in term of job finishing time. This performance improvement on a private OpenStack cloud running on a commodity cluster of 6 compute nodes is measured to be 18%. Nevertheless, in order to show how SETSA can improve performance in a real-world commercial HPCaaS environment, experiments in a larger scale are required. Therefore, we evaluated our implementation of ASETS running SETSA on Amazon public cloud.

### 7.3.3. ASETS on Amazon Public Cloud

Amazon AWS enables us to evaluate ASETS on a larger scale and on an inherently multi-tenant environment. Nevertheless, since access to the hardware and networking infrastructure of Amazon cloud is limited, we need to deploy our implementation of the cloud integrated with an SDN controller. Although this will result in an extra virtualization overhead, we will be able to evaluate the scalability of ASETS and SETSA.

In our experiment, we define the scale of the system by the number of virtual machines launched as the workers. SETSA is expected to perform better as the scale of the system goes up. Results confirm our assumption. Figure 12 shows how SETSA improves the performance of HPCaaS on public amazon cloud significantly up to 67%. As we scale up the system, number of network links and available bandwidths increase, letting SETSA to have a larger variety of choices to redirect data.
Figure 22. The performance of SETSA on Amazon Public Cloud
CHAPTER 8. CONCLUSION

This thesis addresses the problem of multi-tenancy of the network by proposing A SDN-Empowered Task Scheduling System (ASETS) for HPCaaS in the cloud as well as a novel task scheduling algorithm (SETSA) that utilizes SDN APIs to monitor network properties in the cloud for better scheduling. Ideas for improving the performance of SETSA are also proposed by introducing SETSAW (SETSA Window). Our studies confirm previous findings regarding the current challenges for HPCaaS. We identified virtualization overhead, cloud networking, and multi-tenancy as the primary shortcomings of HPCaaS that need to be mitigated.

Previous task scheduling algorithms have not considered the network bandwidth property when assigning tasks to the virtual machines as workers. This is very crucial in the multi-tenant environment of the cloud for HPCaaS where network bandwidths are subject to unpredictable instability. Our empirical analysis indicates that SETSA has the potential to significantly improve HPCaaS in term of turnaround time up to 75% by better utilizing the network bandwidth.

The thesis comprehensively analyzed performance of the proposed task scheduling scheme for HPCaaS; ASETS (A SDN Empowered Task Scheduling System), alongside with its primary scheduling algorithm; SETSA (SDN Empowered Task Scheduling Algorithm). ASETS benefit from a Software-Defined Networking capabilities by leveraging a SDN controller in the architecture. SETSA utilizes the SDN controller to actively monitor the available network bandwidths in order to redirect data over the most suitable link. Our studies and experiments identified three primary challenges for HPCaas; cloud networking, cloud multi-tenancy, and the virtualization overhead. SETSA aims to improve the performance of scheduling data-intensive
HPC tasks on the cloud in term of job finishing time by better utilizing the network. Analyzing the performance of SETSA on two implementations of ASETS on a private OpenStack cloud and Amazon public cloud indicates that SETSA is capable of improving the performance of HPCaaS significantly up to 67%. ASETS is a configurable, dynamic and scalable architecture capable of adapting other scheduling techniques for HPCaaS that may utilize Software-Defined Networking features as well.
CHAPTER 9. FUTURE WORKS

Proposing ASETS as the SDN enabled task scheduling architecture opens doors to a wide area of research and development. This thesis utilized the “bandwidth awareness” capability of Software-Defined Networking in better scheduling data-intensive HPC tasks on the cloud. Other capabilities of SDN in addressing the shortcomings of the Cloud for HPC applications are yet to be investigated.

Although our experiments showed a low overhead for SETSA scheduling algorithm, the algorithm can be improved in performance by assigning multiple tasks to virtual machines at a time. This can be achieved by parallelizing SETSA and considering a window of tasks as the input of the algorithm. SETSA Window (SETSAW) is proposed as a parallel version of SETSA. The performance evaluation of SETSAW and measuring its overhead are parts of the future potential works of this thesis.

Centralized controller of SDN acting as a single point of failure is one of the challenges of Software-Defined Networking. Researches are proposing a distributed Control Plane for SDN to improve its reliability and performance [73]. Accordingly, ASETS also has a potential to adapt to a distributed SDN control plane by providing a distributed task scheduler with distributed tasks queues. Careful analysis of the problem to propose a distributed ASETS is another future research direction.

SETSA schedules tasks based on data transfer time and virtual machine availability. Nevertheless, heterogeneous virtual machines as workers will add more conditions to scheduling strategies of SETSA. Accordingly, SETSA potentially can consider the virtual machines
configurations and costs as extra conditions and schedule tasks accordingly to improve the resource utilization and performance.
REFERENCES


[34] M. Taifi, A. Khreishah, and J. Y. Shi, “Building a private HPC cloud for compute and

and link aggregation on a PC cluster with ethernet,” in IEEE International Conference on

453, 2011.

scientific applications on HPC environments and clouds,” ArXiv Prepr. arXiv1412.6392,
2014.

[38] Bruno Nunes Astuto, Marc Mendonca, Xuan Nam Nguyen, Katia Obraczka,
Programmable Networks. Communications Surveys and Tutorials", IEEE
Communications Society, Institute of Electrical and Electronics Engineers (IEEE),
2014,16 (3), pp. 1617-1634

Patneyt, R. Subrahmaniam, C. Truchan, and M. Tatipamula, “StEERING: A software-
deferred networking for inline service chaining,” in IEEE International Conference on

[40] H. Kim and N. Feamster, “Improving network management with software defined

16, no. 1, pp. 2–14, 2013.


Zagorodnov, “The eucalyptus open-source cloud-computing system,” in Cluster
Computing and the Grid, 2009. CCGRID’09. 9th IEEE/ACM International Symposium on,


