5-2018

Disk-to-Disk Data Transfer using A Software Defined Networking Solution

Junaid Zulfiqar
Clemson University, junaid.rao21@gmail.com

Disk-to-Disk Data Transfer using A Software Defined Networking Solution

---

A Thesis

Presented to

the Graduate School of

Clemson University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Computer Engineering

---

by

Junaid Zulfiqar

May 2018

Accepted by:

Dr. Kuang-Ching Wang, Committee Chair

Dr. Harlan Russell

Dr. Ronald Gimbel

Abstract

There have been efforts towards improving the network performance using software defined networking solutions. One such work is Steroid OpenFlow Service (SOS), which utilizes multiple parallel TCP connections to enhance the network performance transparently to the user. SOS has shown significant improvements in the memory-to-memory data transfer throughput; however, it's performance for disk-to-disk data transfer hasn't been studied.

For computing applications involving big data, the data files are stored on non-volatile storage devices separate from the computing servers. Before computing can occur, large volumes of data must be fetched from the "remote" storage devices to the computing server's local storage device. Since hard drives are the most commonly adopted storage devices today, the process is often called "disk-to-disk" data transfer. For production high performance computing facilities, specialized high throughput data transfer software will be provided for users to copy the data first to a data transfer node before copying to the computing server.

Disk-to-Disk data transfer's throughput performance depends on the network throughput between servers and disk access performance between each server and its storage device. Due to large data sizes the storage devices are typically parallel file systems spanning multiple disks. Disk operations in the disk-to-disk data transfer includes disk read and write operations. The read operation in the transfer is to read the data from the disks and store it in memory. The second step in the transfer is to send out the data to the network through the network interface. Data reaching the destination server is then stored to the disk. Data transfer is faced by multiple delays and is limited at each step of the transfer. To date, one commonly adopted data transfer solution is GridFTP developed by the Argonne National Laboratory. It requires custom application installations and configurations on the hosts. SOS, on the other hand, is a transparent network application without special user software. In this thesis, disk-to-disk data transfer performance is studied with both

GridFTP and SOS.

The thesis focuses on to two topics, one is the detailed analysis of transfer components for each tool and the second part consists of a systematic experiment to study the two. The experimentation and analysis of the results shows that configuring the data nodes and network with correct parameters results in maximum performance for disk-to-disk data transfer. The GridFTP, for example, is able to get to close to 7Gbps by using four parallel connections with TCP buffer size of 16MB. It achieves the maximum performance by filling the network pipe which has 10Gbps end-to-end link with round trip time (RTT) of 53ms.

Dedication

I dedicate this thesis to my parents and family for their love, motivation and support during my studies.

Acknowledgments

I would like to thank my advisor, Dr. Kuang-Ching Wang, for all the help, support, and guidance throughout my studies. Dr. Wang was very helpful and provided valuable feedback and direction towards my research. I am honored to have had multiple opportunities to present demos and talks at conferences/workshops.

Next, I would also like to thank Dr. Gimbel and Dr. Russell for their help and guidance for my thesis. Additionally, Dr. Russell's course work helped me to understand various networking concepts in a much greater, in-depth manner, which was very critical for my research. I also really appreciate Dr. Gimbel for giving me the chance to work with people from a different field. Their input helped me to broaden my networking horizon.

Finally, I would like to thank my colleagues from the research group for their help during my research and experimentation, including Dr. Ryan Izard, Geddings Barrineau, Khayam Anjam, Qing Wang, Casey McCurley and Caleb Linduff. I would also like to thank Kirk Webb, Will Robinson, Robert Ricci and the entire CloudLab community for their help and support towards my experiments.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent times, the workloads and the compute nodes are moving away in the network from the actual user. Thus, the user has to fetch the data from remote locations in order to perform any computation operations or, in some cases, send the data to compute nodes from the local storage. Ideally, a user will require the data to be available to the application on demand, however, the data transfer from/to remote locations involve multiple factors resulting into a significant delay in the data transfer. This thesis introduces and discusses each component of the transfer along with all the factors involved in the disk-to-disk data transfer.

As discussed above, in many cases the need to transfer data is there, however, data transfer is not an instant operation and has to pass through multiple steps. Most of the times data sets are very large and are stored in file systems, thus the first step is to read the data from file system and add it to the network interface buffer. From there the data will be transferred to the destined remote node incorporating network and buffer delay at the receiving end before being available to the application. There have been multiple efforts towards maximizing the disk-to-disk transfer and few client-server tools have been introduced. Tools like GridFTP [1] addresses the disk-to-disk transfer limitations and improves the transfer by using concepts like multiple parallel TCP connections. Users can take advantage of such tools by installing the client and server applications and transferring the data using it[2, 3, 4].

However, these applications do not leverage the new emerging concepts and technologies. One such concept is Software Defined Networking [5] which can be used to remove some of the dependencies of these applications. Specifically, introduction of Software Defined Networking (SDN) has

enabled multiple efforts towards the development of network applications utilizing SDN concepts and increasing network performance and abilities. Steroid OpenFlow Service is one of those network applications and it leverages the SDN to intercept and manipulate the traffic within the network. Earlier studies and implementation of SOS has shown the increased network performance for a memory-to-memory data transfer.

1.1 Data Transfer Solutions

Tools solving disk-to-disk transfer problem includes Globus GridFTP [1] and Aspera [6]. Each one of them has their own benefits and improvements in the data transfer. However, GridFTP boosts the performance of file transfer protocol for high speed data transfer and is a well-known application for large data transfers.

In order to get the GridFTP working, it requires setting up an application on the end user machines. The installation process also includes configurations at the kernel level in order to maximize the throughput performance. The basic setup of GridFTP is based on a client and server model implementation. The server exposes the file system to the client which can read and write to it over the network. Users can use a variety of network configuration parameters as part of the GridFTP. The performance parameters for data transfer includes buffer size, number of parallel streams and block size. Buffer size specifies the TCP buffer size to be used for the file transfer protocol whereas block size is the buffer size for the underlying transfer methods which in this case is block of data to be copied from disk to network. Number of parallel streams is self explanatory and identifies the number of parallel connections to be used for data transfer. Using the combination of buffer size, block size and number of parallel connections, maximum throughput for data transfer can be achieved.

On the other hand Steroid OpenFlow Service (SOS) has been proposed as a network application which leverages software defined networking concepts and transparently intercepts the user traffic. One novel feature of SOS is that once the traffic is intercepted, it can be manipulated by the SOS agents present in the network. More specifically, researchers have shown increased network throughput after the intercepted data traffic is transferred using multiple parallel TCP connections between the agents [7][8][9]. SOS requires setting up an SDN controller with SOS modules [10] and SOS agents [11] at each site. The user does not install any software or application on the end host machines. However, SOS agents are part of the network and must be installed within the network

by network operators. Network operators are also required to modify the controller for enabling SOS connections which will be intercepted in the network and handled through SOS (agents). SOS is also a scalable service and can support multiple users. SOS can boost the traditional network throughput and can also work on top of existing transfer solutions.

1.2 Problem and Objectives

Efforts to speedup disk-to-disk data transfer have been carried out in the past and have shown improvements with custom tools and applications. Similarly, introduction of Software Defined Networking has also sparked efforts towards improving the network performance and limitations including network throughput. Specifically, tools like GridFTP[1] and MPTCP[12] are used to speedup the disk-to-disk transfer which is achieved by configuring the end user machines. SDN solutions like Steroid OpenFlow Service resides in the network as an application and tackles the windowing problem with TCP using multiple parallel connections. SOS doesn't require any configurations at the end user machines as it intercepts the traffic within the network transparently. Data transfer tools are not taking advantage of the new emerging technology tools. In this thesis data center to data center transfer is discussed consisting of transferring data from source parallel file system disks to a remote data center's parallel file system disks. The two data centers utilize 10Gbps end-to-end link with a round trip time of 53ms.

The objective of this thesis is to study the performance along with the dependencies of a disk-to-disk data transfer using GridFTP and Steroid OpenFlow Service. In this thesis, disk-to-disk data transfer throughput is measured for GridFTP and SOS by using different values for parallel connections and buffer values. Specifically, maximum throughput performance for each, GridFTP and SOS, which are about 67% and 60% of the network link are discussed and analyzed in later sections of the thesis.

Chapter 2

Background

2.1 Large Data Transfer Example

Non-volatile setups such as data centers and public clouds have enabled the users to use remote nodes for storage and in some cases as computation resources. If the storage and compute nodes are at the same location, then the delay for data transfer is not very significant. However, if a user is storing the data at one location but it is being generated at a different location, then data transfer faces significant delays. Similarly, in the case of computational nodes on remote locations, it also requires the data to be transferred to the compute location. One such scenario is faced by Genomics researchers discussed below.

2.1.1 DNA Files in Genomics

One particular case of disk-to-disk data transfer is faced by the researchers from Genomics department. In their case, the DNA files which are of the sizes of hundreds of GBs are recorded at DNA collection sites and then transferred to the researchers for computations. The DNA file sizes can increase drastically because of the periodic sequencing [13] and thus require transferring large data files to users. Most researchers from data sciences and analytics are not experts in computing technologies and transfer large data files using traditional tools like FTP on commodity networks which has very poor performance for large data file transfers. Even with transfer tools researchers are not able to achieve maximum performance because of complex configurations on end user machines.

2.2 Parallel File Transfer Tools

Data files can be transferred over local area networks reliably with rapid speeds using TCP. However, link latency and large delay bandwidth networks over wide area networks introduces problems with TCP and is not able to utilize maximum network performance. Multiple solutions have been introduced to tackle the slow data transfer rates in the network with most of them focused on solving the windowing mechanism in TCP connections. The solution tries to solve the issue by filling the pipe of large delay-bandwidth networks. It is achieved by instantiating multiple parallel connections between two nodes. Multiple parallel connections fill the pipe to maximize the network performance and have the data file transferred in parallel to the destination. MPTCP[12] and GridFTP[1] are well-known data transfer tools which use multiple parallel connections to transfers the data.

Users are required to setup the parallel file transfer tools on the host machines in order to benefit from such tools. In most cases, installation of file transfer tools is also required on remote site as well for it to work and give maximum performance. Installation process of parallel file transfer tools also require system level configuration on host which also include modifications in kernel. Researchers with very little or no computing background find it very difficult to configure nodes for optimum performance. This is also because of poor documentation for these tools. It can be concluded that removing the dependency of custom tool installation will be very beneficial for researchers from every field.

2.3 Steroid OpenFlow Service

Introduction of Software Defined Networking has enabled researchers to restructure networks and solve multiple network problems. Researchers are also trying to take advantage of SDN concepts to remove the network as well as some host limitations. Such as in the case of Steroid OpenFlow Service, it is promising to remove the custom software installations from the two hosts participating in a transfer. SOS resides in the network and intercepts the data traffic within network which is enabled by the concepts of SDN. Subsections below discusses the details about SOS.

2.3.1 Design and Architecture

Steroid OpenFlow Service is an SDN based solution and require few components in order to perform the transfer fully and efficiently. The main and most important component is SDN controller which

has the SOS control and operational code, the second one is the SOS agents which are similar to compute nodes and act as the proxy between client and server. The last component is SDN enabled switches present in the network path of the client and server. Each of the SOS component is further discussed in the following sections.

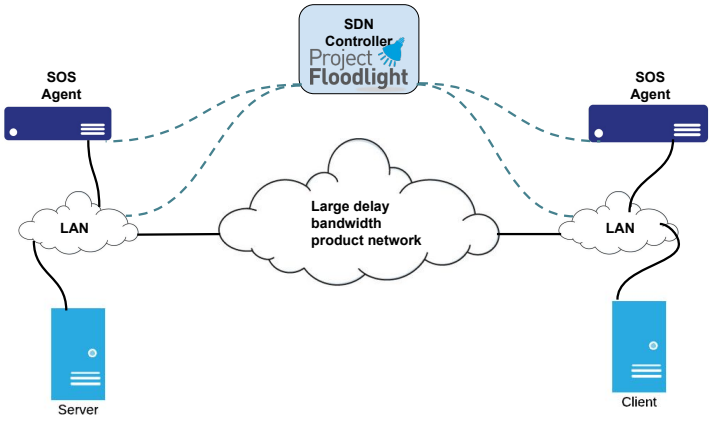

Figure 2.3.1: SOS Architecture

Figure 2.3.1 represents a general SOS setup comprising of two SOS agents, client and server, which are connected through switches and large delay-bandwidth network. SDN enabled switches could be present within the local area network or in the large delay-bandwidth network. Ideally, the switches should be close to the client and server for the best performance. Once the setup is complete and the client tries to establish the connection with server, the packet is intercepted at the SDN enabled switch and sent to the SDN controller for data plane decision. The SOS enabled SDN controller checks for the SOS enabled connection list and if the incoming packet is part of it, then the transfer is started using SOS. Agents are selected and traffic from client/server is redirected to agents which instantiates multiple parallel TCP streams with each other and transfer the data with maximum throughput by filling the pipe. The agents acts as a proxy for client as well as server and maintain a single TCP connection with them transparently without the knowledge of user. Hence the data is transmitted from server to client using multiple parallel TCP streams without having the user to install or configure anything on the client and server.

2.3.2 Agents

An agent is a very important component of the SOS and runs as a C-based application. Agents are responsible for acting as a proxy for client and server, maintaining the TCP connections with them. Additionally, the agents buffer the data before sending and after receiving at server and client side respectively. The data transfer between the agents is achieved by using multiple parallel TCP connections among themselves. The number of parallel connections along with client/server and pairing agent information is sent from the SDN controller as part of the configuration.

2.3.3 OpenFlow Controller

SOS controller is based of Floodlight which is an OpenSource SDN controller. It handles the SOS operation including packet rewrite and redirection flows at SDN switches. It also keeps check of the ongoing SOS enabled transfers along with providing an interface for the users to configure SOS white-listing and usage.

The normal operation for an SOS connection involves capturing the traffic at SDN switches which is sent to the SDN controller. The controller checks the connection against a list of SOS enabled connection configured by the network manager. If the connection is not part of the SOS enabled connection then the controller just handles the packet normally - letting the underlying and non SOS traffic pass through normally. However, if the connection is SOS enabled, then the controller first finds agents present near the client and server. Once the agents are selected, the controller then adds redirection flows to the SDN enabled switches to redirect the packets from the client and server to the agents. The controller also pushes the flows for rewriting packets headers for terminating the TCP connections on the proxy nodes. Agent configurations are also sent to the selected agents which include remote agent, client and server information along with buffer size, number of parallel connections, queue size, etc. After the transfer is finished the agents send the termination message to the controller and controller terminates the SOS connection along with removing all the installed flows.

2.4 Cloud Computing

Cloud computing enables users to access on-demand resources from a pool of available resources over the network [14]. Cloud computing has allowed the users to reserve and access any number of

resources required. These resources are available on different price per resource models, however, cloud computing has made access to resources very convenient for the users. Normally, a cloud computing provider will have a pool of actual physical resources setup at their site locations which are maintained by the provider as well. These resources are connected to the network to be available via internet on-demand. Theoretically, this allows the users to have access to unlimited number of resources. One important benefit of the cloud computing is that the users do not have to worry about any resources as they are maintained by the cloud service providers.

There are different models from the cloud service providers[14] namely, Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Each service model provides different level of resource access. For example, IaaS provides control of physical resources to the user along with low level infrastructure including network, computing resources, security etc through APIs. Similarly, in PaaS, the users get access to a full platform such as Linux virtual machine and can run multiple desired applications. In SaaS, a user access is limited to a specific application using a user interface or HTTP client. However, in all the cases, there is a data transfer requirement. Users in most cases need to transfer the data to the remote nodes hence increasing the potential use of tools like GridFTP and SDN based solutions to provide higher throughput.

With the increase demand of computing resources, the workloads are growing very fast. This workload growth introduces the demand for increased throughput as well. Only scalable data transfer solutions can provide required throughput needs. SOS has a scalable architecture and can use different SOS agents to handle multiple data transfer requests.

Chapter 3

Disk-to-Disk Data Transfer Analysis

Large data transfer to or from remote locations can be broken down into components. Mostly large data sets are stored in storage disks, hence the first step is to read the data from storage disks and save to memory. The next step is to move the data from memory to the network interface buffer and once there is enough data in the interface buffer, transmit it to the network. Similarly on the receiving end data will go through the same process. These steps are discussed in detail below to identify the expected bottlenecks in the disk-to-disk transfer.

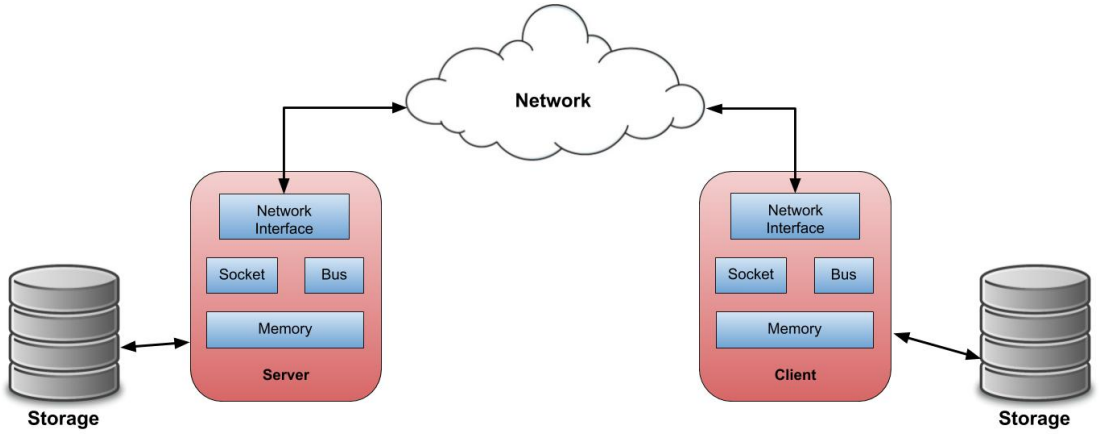

Figure 3.0.1: Disk-to-Disk Data Transfer Components

3.1 Storage Throughput Analysis

The first step of the data transfer, which is also the slowest part of the transfer, is reading the data from disks even if the destined application is running on the same host. The potential bottlenecks which can limit the transfer can be broken down into following important components [15].

- Disk bandwidth
- Disk drives
- Disk controller
- Cache
- Non-uniform memory access (NUMA)

Different factors from these account in for every different case. Disk I/Os are usually the main limiting factor because of the advanced disk controllers such as SATA [16] and RAID. Classic disks cannot perform very well when multiple applications try to read the data from the disk and do not have sufficient number of drives for better throughput. Hence, the traditional disks such as spinning disks become the bottlenecks when these advanced controllers are used. Controllers like SATA III can give transfer rates of the scale of 6Gbps per port as compared to 500Mbps for spinning disks

[17]. However, disk controllers can also become bottlenecks in some cases. Factors like overloaded CPU and less processing power can limit the controller from achieving maximum performance. Also, if advanced disks are used, then the controllers cannot keep up with the performance of disks. For example, each solid state drive (SSD) can achieve about 4Gbps and most of the aggregate controllers are capable of achieving about 10Gbps. Hence, the controller can easily become bottleneck by using multiple SSDs in the setup. Using multiple controllers can remove the controller bottleneck, however, CPUs have limited peripheral component interconnects (PCI) for disk controllers and cannot resolve the controller bottleneck in that case.

In case of multiple controllers, applications will need to be aware of the NUMA architecture and access the data from specific CPU sockets on both source and destined host otherwise CPU will use local bus for transferring the data. This could also be a potential bottleneck. Disk controllers also have cache which is beneficial for the disk read and writes. However, the cache is limited in size and can only provide better performance as per the size. The cache can also be a potential bottleneck if it is overloaded with disk write operations and if non-sequential data is used which results in cache rewrites frequently thus adding potential delays.

Single disk limitations can also be eliminated by having multiple disk storage nodes. Such solutions have multiple disk storage nodes in parallel which are used to store the data in a hierarchical order. The parallel storage controller receives the data and then stores it onto parallel disks in a rotating manner. The client for parallel system can have single controller or multiple depending on the requirements and desired performance. Common examples of such systems are BeeGFS [18] and OrangeFS[19] which maintains the parallel storage nodes from the client node. These file systems represent the files as mount points on the client's local system but actually they are stored in the parallel files system. A client can read and write to the file through these mount points. When a client tries to read or write to the file mount point, the file system controller translates the operation to collecting or storing the data to the parallel nodes depending on the operation. The maximum performance is achieved when a separate network interface card is used to read and write from the parallel file system disks.

3.2 Server Throughput Analysis

Once the data is out of the storage, there are factors within the server which play important part in the data transfer. Some of the important limiting factors in the data transfer are mentioned below.

- Memory
- Socket
- PCI Bus
- Network
- CPU

Memory is very important component in the disk-to-disk transfer and memory read/writes have to be efficient in order to achieve maximum performance in the transfer. With the introduction of modern high performance CPUs the maximum performance is achieved with minimum memory accesses which can achieved by increasing the size of cache[3]. However, the increase in number of applications requiring different data from the memory introduces cache misses and needs a very complex algorithm to avoid them. Hence, the performance of multi-processor system will be worse with single memory accesses. NUMA solves this problem by introducing separate memory for each processor to increase its performance [20]. The bus also plays important role in the NUMA architecture and if the applications are not connected to the same CPU processor or bus, then they will need to fetch the data from shared or non-local memory (which is local to the other processor). Hence, if the storage controller and application memory are not connected to the PCI buses on the same CPU processor, then it will cause a miss and introduce delay. Similarly, if the application and network interface cards are not on connected to same CPU, it will also cause a miss and introduce latency in the data transfer. In the scenario with a parallel file system, the client (controller) reads the data from the storage disks. The data is fetched to application memory by using a separate network interface. Similar to the other case the storage controller application and the network interface should be connected to same CPU in order to receive the data in the memory without any significant delays.

Once the data is in memory, the next step is to push the data to the network by using sockets. Data is written to a socket by the application and the system kernel can transfer the data from

there to the network interface. When writing the data to the socket, the application can either copy the data from user space to kernel space buffers or it can use memory-mapped buffer allowing the kernel to directly access the data from memory. However, this adds buffer management complexity for the application. Afterwards, the data is transferred from kernel to the network interface card either using direct memory access or CPU. As mentioned earlier, in order to achieve maximum performance, the network interface card and application memory should be connected through same bus and CPU processors to avoid delays.

3.3 Network Throughput Analysis

Data from storage devices passing through the memory reaches the network interface card is transmitted to the network towards destination. However, there are various factors in the network which can introduce bottlenecks and result in delays. The network limitations range from network congestion to transfer protocol configurations. If the underlying network is not using network components, such as routers or switches, to handle the incoming traffic from high speed connections it will introduce network congestion and affect the network throughput. However, the modern network switches are still limited with the amount of memory they have and can only buffer a limited amount of packets waiting to be processed. Hence, the packet drop event is unavoidable with current setups.

Another limiting factor is the windowing mechanism in the TCP – which is mostly used for data transfers. The TCP keeps a window size based on which it transmits the data packets and this window size is increased with the packet ACKs and receiver's advertised window size. However, with the expected packet drops, the window size is reduced drastically and doesn't fill the network pipe to get maximum performance. Hence, solutions like MPTCP [12] have been introduced which try to solve the windowing problem in TCP by starting multiple parallel connections. With this, every parallel connection keeps its own window size and even if one of the connection experiences a packet drop it will change the window size of its own. In this way all the other connections doesn't know about the packet drop and keep their window sizes higher which helps in filling the network pipe and getting maximum network performance.

3.4 SOS Throughput Analysis

Steroid OpenFlow Service is integral part of disk-to-disk transfer discussed in this study. It's better performance in network has already been shown in separate studies [8, 21, 15]. However, there are multiple factors which are to be studied in order to achieve maximum performance from it. Some of the limiting factors which can be considered as bottlenecks include:

- Data movement
- Memory
- CPU Performance
- Buffer size
- Data sequencing

With the introduction of agents in the network and redirection of traffic from the switch can require the data to traverse additional links. However, depending on the presence of agents with respect to client and server the expectation of performance can be set. For example, if the agents are present very deep in the network path of client and server then the multiple parallel connections will not really improve the performance as compared to single connection. It is because of the fact that most of the network path is using single connection between the agent and the client or server. On the other hand better performance is expected when the agents for SOS are located near the client and server because most of the network is covered by multiple parallel connections between SOS agents. Hence, the presence of agents in some cases can also pose limitation in the disk-to-disk data transfer.

Once the data reaches an agent it is forwarded to the peered agent over parallel connections. Agents are not meant to store the data traffic but only to act as a network forwarding device. However, in order to send the data chunk of desired size, the sending agent might have to store the data in memory. Similarly, the receiving agent might have to store the data in buffer in proper sequence to take care of packets arriving in a non-orderly way due to TCP connections taking different paths. In both the cases the data needs to be stored in the agents memory, thus it can be a bottleneck for transfer if the memory size is small as compared to the data needing to be stored. This can be avoided by configuring the agent nodes with large amount of RAM. As mentioned in the above sections moving the data from network to memory can also introduce a bottleneck but

it could be reduced to negligible by using direct memory access which enables the network card to directly access memory for read/writes.

An agent node must be equipped with a CPU which can handle receiving data from the network and sending it out efficiently. Also, the network card and the agent application should be connected to the PCI bus on same CPU for maximum performance. It allows the network and application to share the data faster and minimize the limitation.

Chapter 4

Experimental Studies

Pair of client and server is required for the experimental setup of disk-to-disk data transfer using software defined networking solution. The experimental data is also required on the client along with full SOS setup in the experiment. The SOS setup includes SOS agents, SOS controller and SDN enabled switches in the network path. In most cases researchers need to transfer a number of files which account for large sizes and are stored in file systems. For example, mostly DNA genomics data is stored in parallel file systems. Thus, to make the study and experiments more realistic parallel file systems are also included in the experimental setup. Further details of the experiment topology are discussed in Section 4.2.

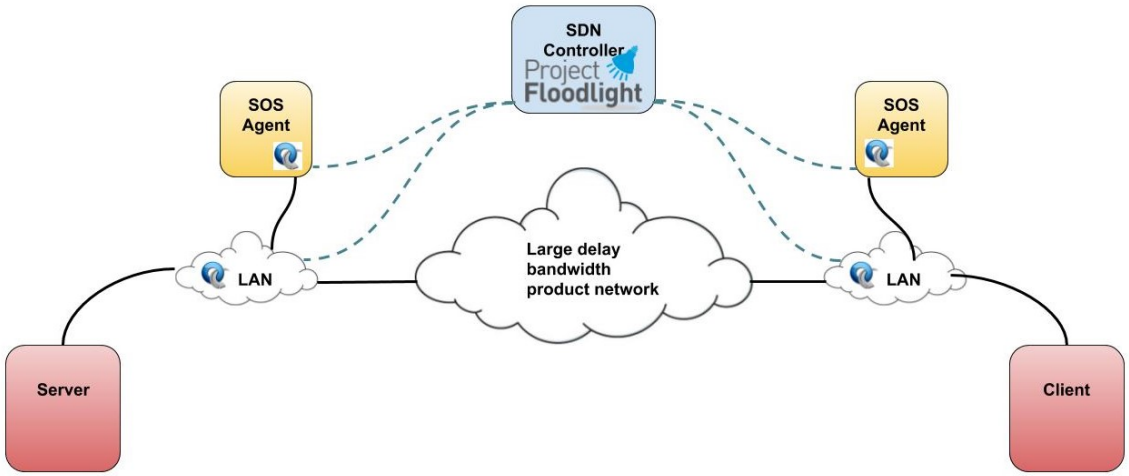

Figure 4.0.1: Disk-to-Disk Data Transfer Topology

4.1 Experiment Design

Disk-to-disk data transfer includes experiments using GridFTP and SOS. In order to study the disk-to-disk data transfer components and performance, sets of experiments involving large data file transfers are required. The basic experiment for data transfer requires client and server nodes connected over network. To study performance of transfer tools over large delay-bandwidth networks, the server and client should be connected using a large delay-bandwidth network. Imitating real data transfers require actual experimentation data and it should be used for all experiments. The first experiment consists of gathering results using GridFTP varying number of parallel streams and buffer size as baseline. The next set of experiment includes disk-to-disk data transfer using a file transfer application over SOS. The next experiment comprises of integrating SOS with any other fast data transfer tool (e.g. GridFTP) to study the working of SOS with already present data transfer tools. Lastly, experiment involving multiple agents and multiple data transfer streams to discuss the transfer at scale.

4.2 Experimental Setup

A single topology similar to Figure 4.0.1 can be used for all the experiments discussed in the Section 4.1. However, to store large data files and better throughput, a parallel file system has to be introduced as well. Most of the file systems can be setup with extra storage nodes connected and controlled from the client. Hence, adding desired number of storage nodes is enough to complete the setup. In this particular experimental setup, eight file system nodes are added connected to client and server. The client and server of the data transfer in Figure 4.0.1 can be used as the file system clients by using a separate network interface facing towards the file system storage nodes.

Experimental topology spanned over two sites having compute resources at Clemson University in Clemson, SC and University of Utah at Salt Lake City, Utah. The connection between Clemson and Utah utilizes AL2S setup which can provide 100 gigabit Ethernet between different sites. However, the interconnects between the nodes within each site are limited to 10G hence, providing 10Gbps for disk-to-disk connection. All the resources are provided and hosted by CloudLab which is an NSF supported SDN based cloud provider [22] and can be used for running or testing experiments. Details about the resources and setup are discussed in following sections.

4.2.1 SOS Setup

The main part of this study involves observing the disk-to-disk transfer using Steroid OpenFlow Service. Therefore, the most important component of the experimental setup is SOS. For this set of experiments, multiple SOS agents discussed in earlier sections are setup along with client and server spanned over two sites. Eight compute nodes are installed and configured with SOS agent applications at both sites with each having four agents. To clarify, configuring and setting up agents is part of the network operators which can be utilized by users.

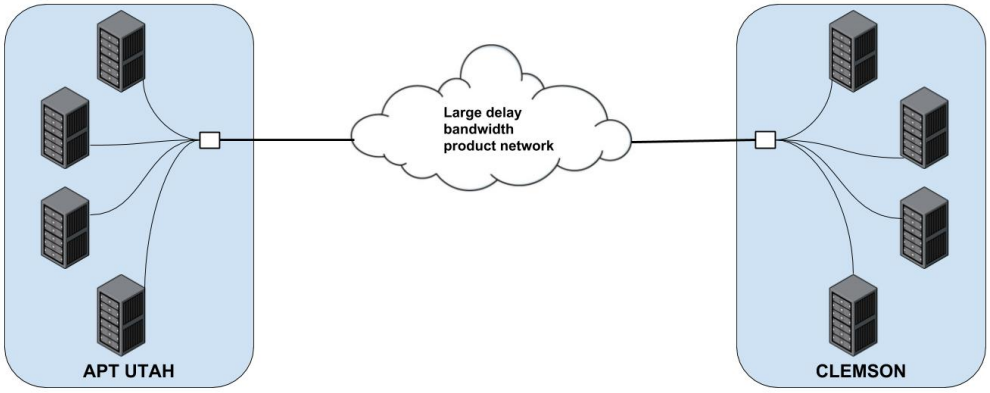

Figure 4.2.1: SOS Agents on Clemson and Utah

All the agents are connected to SDN controller which is running on a Ubuntu machine at Clemson OpenFlow lab. The SDN controller is equipped with SOS modules and is responsible for maintaining and carrying out SOS enabled connections. The controller can also handle the normal or non-SOS traffic and lets the traffic to go through the network without any redirection and rewrites. Configuring the controller with SOS modules and SOS connection is also part of the network level configurations. The controller has to be configured with the expected SOS connection in advance so that the controller will transfer the data using SOS.

The controller receives the traffic from OpenFlow enabled switches and checks if it is configured as an SOS connections. If the controller find the connection in a configured list of SOS connections, it looks for SOS agents present near the end hosts. Once it finds the agents, it sends the configurations to the selected agents which includes the information about number of parallel connections, buffer

size along with remote agent. The controller also push flows to the OpenFlow enabled switches to rewrite the packet headers for redirecting them to the agent and vice versa. The agent-to-agent traffic of multiple parallel TCP connections is handled normally in the controller as well as the network.

The whole packet walk in SOS comprises of multiple steps. Firstly, once the controller selects the agents and pushes the flows to the switches, the incoming traffic from client is redirected to the client side agent by modifying the packet header information. The packet data is stored in a memory buffer when the packet reaches the agent. The agent either waits for the buffer to be filled up to a certain configured size or timeout, after which it transmits the data to the remote agent using already established parallel TCP connections. When the remote site or server side agent receives the data, it also stores it in the buffer and sends it to the server (achieved through rewrites and redirection) on the single TCP connection. The server receives the data packets with modified headers and considers the traffic to be coming from the client. Hence, the parallel transfer is done only between the agents and transparent to the user.

Few experiments also involved multiple agents at each site. For those experiments the client and server initiated two parallel connections which were handled by two agents on each site separately. In this scenario the first stream of data is handled the same way as described in the above discussion. Similarly, the second stream is intercepted at the switch and the controller sets up separate agents to handle the stream. The two data streams from client and server are terminated into the two selected agents on each site and the data is transferred across sites using parallel connections between the agents. Hence, the SOS setup is able to handle multiple streams of data due to its scalable architecture and using multiple agents.

4.2.2 Parallel File System

As discussed in the earlier sections, big data and data computing have introduced increasing number of data files. Also, in this experiment the size of data files used is 862,506,158,915 bytes. In order to store and achieve better throughput for large data files, multiple parallel disks are required which are best provided by parallel file systems. Multiple parallel file systems have been introduced depicting better performances with multiple disks. Few parallel file systems have already been tested for their performance with respect to read and write throughput [13]. Particularly, BeeGFS and OrangFS [19] outperformed other file systems with BeeGFS showing best read throughput and third best

write throughput and OrangeFS depicted second best read throughput and best write throughput. Further, BeeGFS outperformed OrangeFS in a transfer of actual data set. BeeGFS showed consistent throughput of more than 270MBps when transferring data file with 4 TCP streams [13].

CloudLab provided two network interface options for file system read/writes. One of which is to use same 10G TCP link which is shared with data traffic going to the destination and is not desired. The other option is Infiniband, which theoretically can outperform TCP 10G link with 40G. It is also verified that Infiniband shows better performance for higher number of TCP streams for a full transfer. Infiniband is able to achieve 40% higher throughput with 16 streams as compared to TCP maximum throughput at 13 streams. Therefore, Infiniband is used as the network interface communicating with the parallel file system nodes.

4.2.3 GridFTP

BeeGFS and other well-known file systems represents the data files as single mount points in client local system. Although the files in parallel file systems are divided into data chunks and stored in different storage nodes. So, any application is able to read and write the data using the file mount point on client. For this experiment multiple applications were used, however, most of them did not perform very well because of poor handling of reading data files from the PFS and were not able to transmit the data to the network efficiently. Due to the poor performance of native applications, it was decided to use GridFTP with one and multiple parallel streams to transfer data files using multiple SOS agents respectively. GridFTP can handle the read/writes from the file system very efficiently and can send the data traffic rapidly to the network.

4.2.4 CloudLab Topology

The experiments topology was setup in CloudLab which is an SDN enabled testbed and allows researchers to deploy experiments on the cloud [22]. CloudLab has resources at multiple sites including Clemson, Utah and Wisconsin. Researchers can reserve resources from any site and also connect them with different site which is achieved by stitching the link with a separate VLAN allowing L2 network access within the experiment. Earlier studies [8, 15] had SOS experiments in GENI[23][24] but the network throughput in GENI is very limited and CloudLab provides high speed end-to-end connections up to 10G. Another reason moving to CloudLab was that it also provides

physical resources for the experiments which is very appealing for the SOS agent's performance as discussed in the earlier sections.

Topology shown in Figure 4.2.2 represents the nodes at both Clemson and Utah site. Total 26 nodes are used in the experiment with 13 nodes present at each site. Out of 13 nodes 8 are used in setting up parallel file system using BeeGFS. The BeeGFS client is setup on the file transfer end nodes and rest of the 4 nodes are reserved for SOS agents. Just to note, only two were used in the actual experimentation because of scalability issues with parallel application discussed in later sections. All the resources are connected through 10G links within each site along with a separate Inifiniband network of up to 40G. The connection between two sites is based on high speed AL2S link which provides up to 100G connection.

Details about the resources used in the experiment are discussed in following sections.

4.2.4.1 Specifications

The nodes specification at both Clemson and Utah site are as follow:

Clemson

- CPU: Two Intel E5-2683 v3 14-core CPUs at 2.00 GHz (Haswell – 28 cores)
- RAM: 256GB ECC Memory
- Disk: Two 1 TB 7.2K RPM 3G SATA HDDs
- NIC: Dual-port Intel 10Gbe NIC (X520)
- NIC: Qlogic QLE 7340 40 Gb/s Infiniband HCA (PCIe v3.0, 8 lanes)

Nodes used at Clemson are c6320 which is configured with two Intel Haswell processors providing 28 cores in total. These nodes can hold up to 256GB of RAM and has two 1 TB SATA hard disks. They are also equipped with one 10G network interface which is used for the data network spanning over 100G Al2S link. Infiniband interface is also present which is used to connect file system nodes with the client. In addition to these interfaces another 1 Gbps network interface is used as management interface in all nodes [25].

APT Utah

- CPU: Intel Xeon E5-2450 processor (8 cores, 2.1Ghz)

- RAM: 16GB Memory
- Disks: Four 500GB 7.2K SATA Drives - 1.36 TB RAID0 partition for data transfers
- NIC: 1GbE Dual port embedded NIC (Broadcom)
- NIC: Mellanox MX354A Dual port FDR CX3 adapter w/1 x QSA adapter

r320 APT Utah nodes are used in the experiment setup which are equipped with Intel Xeon processors providing 8 cores and 16GB of RAM. These nodes contains four SATA hard disks with each 500GB in size. The nodes are also equipped with two network interface cards, one of which is 1G link and is used as management interface. However, the other one is Mellanox and has Mellanox's VPI technology which allows the ports on NIC to be configured as Inifiniband, or a 40 Gbps Ethernet. One port on same Mellanox interface card is configured as 10G Ethernet to be used in the data network using Al2S links.

4.2.4.2 Network Connectivity

Data network on both sites are connected with physical OpenFlow enabled switches in the data network. Both sites use Dell Force10 switches and all the nodes are connected with each other using one or more switches at each site. However, while setting up the experiment the nodes connected to any switch other than the egress switch were not able to communicate with remote site after enabling OpenFlow. This behavior was similar to what was seen in earlier experiments as well and is due to the same bug present in Dell firmware. The bug is fixed in Dell's latest releases however, the patches have not been applied to CloudLab setup yet. In order to avoid this issue and working with CloudLab support, it was decided to just use the nodes which are connected to egress switch only.

An OpenFlow instance was created in the experiment setup and configured with SDN controller IP and port. This allowed the controller to create links and control the switches. Switches send the packet-ins to the controller using this OpenFlow link and controller configures the switch by pushing flows of rewrites and redirection. The SOS agents also run a virtual switch for packet rewrites and in the experiment Open vSwitch was used in each agent which is an OpenSource virtual switch. These switches were also connected to the same controller running in Clemson. The connection between controller and virtual switches utilized the management network and hence did not interfered with the data network in the transfer.

4.2.4.3 Infiniband

Along with data network interfaces in a node, it has another infiniband network that can provide a link up to 40G. This link is utilized for setting up the parallel file system storage nodes. The file system client sends the data files using the infiniband link and also uses the same link to read the data from those disks. This allows the setup to have a discrete connection for parallel file system and a separate connection for data traffic. Hence, the file system traffic does not interfere with the data transfer traffic going from client to server.

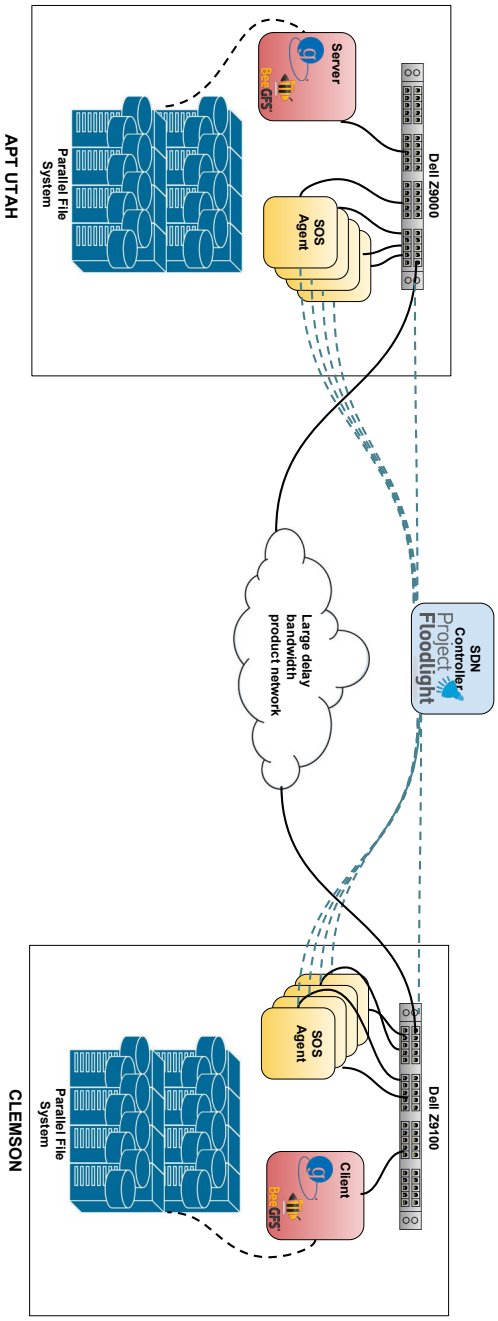

Figure 4.2.2: Experiment Topology

4.3 Results and Analysis

The next step after setting up the experiment topology was to test the working before running the experiments. The connectivity of each node was tested using a simple ping test which verified the ARP handling along with a steady connection. The approximate round trip time (RTT) between both sites was around 53ms on the data network. Once the connection was tested, the experiments were ready to be performed. Each experiment was set to run 10 times however, some of the experiments were only able to run for up to 5 times due to limitations of the agent and the setup which included agent application crashing after few runs. Near the end of the study the experimental setup stopped to communicate between the two sites due to the same Dell bug (explained in Section 4.2.4.2) but for another reason which is still unknown and has not been fixed. This limited the experiments to run for desired number of times.

The results for experiments are divided into following sections.

4.3.1 End-to-End Analysis

The end-to-end analysis included experiments with disk-to-disk data transfer using GridFTP, wget and SOS. The experiments were carried out on the same experimental setup as described and discussed in Section 4.2. The first set of experiment involved full disk-to-disk transfer using GridFTP with different number of parallel streams.

4.3.1.1 Transfer using GridFTP

The disk-to-disk transfer over GridFTP was carried out in order to get the baseline. In this experiment the same topology as shown in Figure 4.2.2 was used with 8 parallel file system nodes at each site. However, the transfer connection was not included into the SOS connection list which is used by the controller to decide whether to handle the connection by using SOS or not. In this case when the data transfer is started and intercepted by the controller, it does not find the connection in SOS enabled connection list. Thus the controller handles the traffic normally and push the forwarding flows to the switches without any redirection and rewrites.

Table 4.3.1 shows the throughput achieved by using GridFTP for a full disk-to-disk transfer. The experiment transferred all the data files of size 862,506,158,915 bytes spanning over different number of parallel connections. The number of parallel connections sweep through 1 to 64 connections with

No. of Streams	Throughput (Gbps)	Standard Deviation	Confidence Interval 95%
1	1.7086	0.0331	0.0374
2	3.5789	0.4127	0.4670
4	6.6004	0.5206	0.5891
8	3.9361	0.2699	0.3054
16	4.4858	0.1570	0.1776
32	4.6537	0.4195	0.4747
64	6.1156	0.3287	0.3719

Table 4.3.1: Average Throughput, Standard Deviation and 95% Confidence Interval for GridFTP

TCP buffer size of 16.7MB. It can be noticed from the table that GridFTP achieves maximum throughput for 4 parallel streams and then drops. It is due to the fact that with 4 parallel TCP streams the transfer is able to fill the pipe.

It is mentioned earlier that the RTT between the two sites is 53ms with end-to-end connection of 10Gbps. Hence, the bandwidth delay product results in 66.25MB and will require the TCP buffer size to match it in order to fill the pipe. However, in the experiment, the TCP buffer size used is 16MB and if four parallel connections are used the data on the network turns out to be more than the bandwidth delay product i.e. ~67MB. Thus, with TCP buffer size of 16.7MB and four parallel streams the data transfer fills the pipe and achieve the maximum throughput in the table. Afterwards, with the increasing number of parallel streams the throughput drops instantly because of increased packet drop count which results in TCP backoff for all the streams. However, when the number of streams is increased, packet are still being dropped with TCP backoff, but there are still enough streams to keep the pipe almost full. Hence, the throughput is increased with increasing number of parallel streams with going above 6Gbps for 64 parallel streams.

4.3.1.2 Transfer using wget over SOS

The second set of experiments involved disk-to-disk transfer using wget [26], which is a file retrieving tool and uses HTTP or FTP protocol to achieve this. However, with this large data set

wget was not able to carry out the transfer even with SOS and performed very miserably. It was because of its poor performance in putting the data over the network from the parallel file system introducing a huge bottleneck in the data transfer. Hence, the experiment was not able to finish and there were no results collected for this experiment.

4.3.1.3 Transfer using GridFTP over SOS

As described in section 4.2.3, the normal file transfer applications were not able to read the data from file system and move it to network quickly for utilizing the high speed network. In order to overcome this limitation of file system read/write and memory management GridFTP was used. In these experiments GridFTP was limited to single stream. Hence, GridFTP only removed the end host limitations but the transfer in network was handled by Steroid OpenFlow Service and its agents.

No. of Streams	Throughput (Gbps)	Standard Deviation	Confidence Interval 95%
1	3.7308	0.4192	0.4744
2	4.8472	0.0816	0.0924
4	5.1724	0.0719	0.0814
8	5.0597	0.0430	0.0487
16	5.0879	0.0893	0.1010
32	5.2692	0.1361	0.1540
64	4.9361	0.0251	0.0284

Table 4.3.2: Average Throughput, Standard Deviation and 95% Confidence Interval for Steroid OpenFlow Service

For the first set of experiments only one stream of data transfer was initiated from GridFTP which was white-listed in the controller to be handled by SOS. In SOS the number of parallel TCP connections also ranged from 1 to 64 in the same manner as for GridFTP. A full disk-to-disk transfer was carried out multiple times for a particular number of SOS parallel connections. Then an average throughput was calculated based on the time calculated for each transfer. Table 4.3.2 shows the average throughput results for 1,2,4,8,16,32 and 64 parallel TCP connections while keeping single

stream from GridFTP to be intercepted by SOS. The table also shows the standard deviation along with confidence interval of 95% over throughput. The throughput is increased as the number of parallel connections between SOS agents are increased.

From the table 4.3.2, it is observed that the throughput tends to increase. However, a drop in the throughput is seen for 64 parallel streams. This is because the buffer size is configured in the SOS agents as 10000 which acts as a bottleneck for large number of parallel streams. Table 4.3.3 shows the average throughput against different buffer size for higher number of parallel streams. It can be seen from the table that using larger buffer sizes for higher number of parallel streams results in better disk-to-disk data transfer throughput. Hence, from Table 4.3.2 and Table 4.3.3, it can be inferred that small buffer size in SOS yields better performance for less number of parallel streams and large buffer size provides better performance for higher number of parallel connections.

No. of Streams	Buffer Size	Throughput (Gbps)
64	10000	4.9361
64	60000	5.7857
64	65000	5.9840

Table 4.3.3: Average Throughput against Buffer Size for Higher Number of Streams

Figure 4.3.1 shows the graph of disk-to-disk data transfer for both GridFTP and SOS. Blue trace shows the average throughput achieved using GridFTP for data transfer using 1 to 64 parallel streams. On the other hand, orange trace shows the throughput achieved over SOS for same number of parallel streams. Clearly, we can see from the graph that SOS is able to almost match the maximum throughput achieved by GridFTP. To be more precise, GridFTP achieves maximum throughput of 6.6Gbps with four parallel connections. SOS is able to achieve 6Gbps for 64 parallel streams which is a great performance value keeping in mind that SOS is introduced as a network application and doesn't require custom installations on the end hosts. It is also shown from the graph in that SOS is actually able to achieve higher throughput than GridFTP in most of the transfer parameters and only lags behind GridFTP for 4 and 64 parallel connections.

All the experiments for disk-to-disk data transfer were carried out using the setup on 10G end-

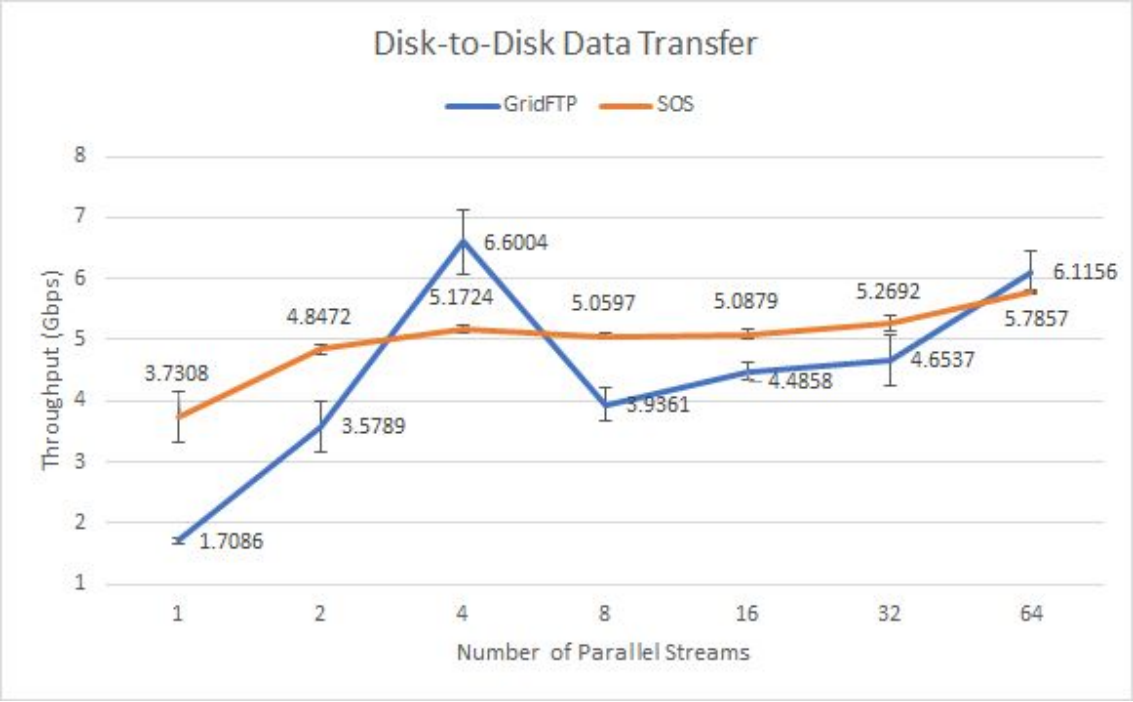

Figure 4.3.1: Disk-to-Disk Data Transfer Throughput using SOS and GridFTP

to-end link. However, the results show the highest bandwidth achieved either with using a parallel file transfer tool i.e. GridFTP and also with using a network application based on Software Defined Networking. Both the approaches could achieve maximum average throughput of approximately 6.6Gbps which is 66% of the total bandwidth link. It shows that the data transfer faces delay when transferring from disk to disk. The delay is due to the limitations and bottlenecks discussed in Section 3 which are present in the network along with the client and server nodes. In addition to all the limiting factors, it is also worth to note that the experimental setup is based on CloudLab which is a very popular tool among researchers and the performance can vary based on the usage of CloudLab network. The experiment runs were carried out at different times of the day to get the performance over range of network usage.

4.3.2 Scalability

The next set of experiments was to run GridFTP with two parallel streams. In this experiment the GridFTP server uses port 40001 and the GridFTP client instantiates two streams originating from port 50001 and 50002. These two streams are already configured in the SOS controller and once the

controller intercepts the traffic from both streams originating from client it selects SOS agents on each site for the first stream which establishes the proxy connections with client on port 50001 and server on port 40001. Next, when controller selects the agents for second stream it sometimes selects the same agent on one side and sometimes different agents on each site which is due to a known bug in SOS controller. Selecting different agents from the first stream is the desired behavior which allows the second stream to be handled by totally different agents in order to achieve maximum performance and scalability. As soon as the agents for second stream receives the configurations from the controller, the client side agent establishes the connection with client on port 50002 i.e. for second stream. However, when the server side agent tries to establishes the connection with server on port 40001 it is not successful because of the rewrites from SOS. This also happens even if same agent is selected on server side. Hence, the disk-to-disk data transfer using multiple agents was not be completed. The fix to this issue is out of the scope of this study and is left for the future work.

The expected performance with using multiple GridFTP parallel streams could be increased theoretically. For example, in the experiment mentioned above, two parallel streams from GridFTP are supposed to be handled by two separate SOS agents. The SOS scalability has already been shown in earlier studies [15, 8] with the throughput increasing linearly with the number of agents. Hence, in this case the throughput is expected to be doubled if both the parallel streams are handled correctly by SOS agents. Similarly, if more than two parallel streams are used in GridFTP which are handled by different SOS agents, the throughput should also increase linearly with the number of increasing parallel streams and will only be limited by the backbone internet throughput which in this case is 100G for AL2S.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

The thesis addresses the disk-to-disk data transfer by using GridFTP and Steroid OpenFlow Service, which is an software defined networking solution. The analysis shows that the throughput of both tools depend on multiple factors. Understanding these limiting factors is critical in achieving the maximum throughput. Limiting factors range from storage and server nodes to the network congestion. Experimental factors include the different number of parallel connections, buffer size, pinning the processes to specific CPU, sockets configuration and network parameters. Increased performance for disk-to-disk data transfer was achieved when limitations were avoided by using a parallel file system, pinning the applications and network interface cards to the same CPU and filling the end-to-end network pipe by using multiple parallel connections along with increased buffer size. To achieve maximum throughput, the bottlenecks for disk-to-disk data transfer should be avoided by proper configurations and setups.

5.2 Future Work

The thesis addresses the potential bottlenecks for a disk-to-disk data transfer and studies the performance with using two tools namely GridFTP and SOS. Accepting this work on bottlenecks and parameters, further studies can be concluded to fine tune the parameters to achieve maximum throughput. There is a need to study scalability and will require modifications in SOS agents along

with the controller to work with GridFTP. Additionally, the SOS agents are not very stable and are likely to crash after few runs, requiring more work towards the stability of the agents. Data transfers can also be studied with other transfer tools to compare and study the limiting factors. It will also show the flexibility of SOS to work with any application.

Bibliography

[1] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link, "The globus striped gridftp framework and server," in *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, Nov 2005, pp. 54–54.

[2] A. Rajendran, P. Mhashilkar, H. Kim, D. Dykstra, G. Garzoglio, and I. Raicu, "Optimizing large data transfers over 100gbps wide area networks," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, May 2013, pp. 26–33.

[3] E. S. Jung, R. Kettimuthu, and V. Vishwanath, "Toward optimizing disk-to-disk transfer on 100g networks," in *2013 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, Dec 2013, pp. 1–6.

[4] S. Liu, E. S. Jung, R. Kettimuthu, X. H. Sun, and M. Papka, "Towards optimizing large-scale data transfers with end-to-end integrity verification," in *2016 IEEE International Conference on Big Data (Big Data)*, Dec 2016, pp. 3002–3007.

[5] K. Benzekki, A. E. Fergougui, and A. E. Elalaoui, "Software defined networking (sdn): a survey," February 2017. [Online]. Available: https://doi.org/10.1002/sec.1737

[6] "Aspera, inc." [Online]. Available: http://asperasoft.com/

[7] A. Rosen, "Network service delivery and throughput optimization via software defined networking," 2012. [Online]. Available: https://tigerprints.clemson.edu/all_theses/1332/

[8] R. Izard, C. G. Barrineau, Q. Wang, J. Zulfiqar, and K. C. Wang, "Steroid openflow service: A scalable, cloud-based data transfer solution," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2016, pp. 447–452.

[9] R. Izard, J. Deng, Q. Wang, K. Xu, and K.-C. Wang, "An agent-based framework for production software defined networks," *Int. J. Commun. Netw. Distrib. Syst.*, vol. 17, no. 3, pp. 254–274, Jan. 2016. [Online]. Available: https://doi.org/10.1504/IJCNDS.2016.080112

[10] "Sos for floodlight." [Online]. Available: https://github.com/rizard/SOSForFloodlight

[11] "Sos agent." [Online]. Available: https://github.com/geddings/sos-agent

[12] M. Scharf and A. Ford, "Multipath tcp (mptcp) application interface considerations," Tech. Rep., March 2013. [Online]. Available: http://www.rfc-editor.org/info/rfc6897

[13] N. Mills, F. Feltus, and W. L. III, "Maximizing the performance of scientific data transfer by optimizing the interface between parallel file systems and advanced research networks," 2017. [Online]. Available: http://dx.doi.org/10.1016/j.future.2017.04.030

[14] P. Mell and T. Grance, "The nist definition of cloud computing," Sept. 2011. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[15] R. Izard, "Data movement challenges and solutions with software defined networking," Ph.D. dissertation, 2017. [Online]. Available: https://tigerprints.clemson.edu/all_dissertations/1910

[16] "Sata." [Online]. Available: https://www.pcmag.com/encyclopedia/term/50811/sata

[17] Y. Lee, H. Yoo, I. Yoo, and I. C. Park, "6.4gb/s multi-threaded bch encoder and decoder for multi-channel ssd controllers," in *2012 IEEE International Solid-State Circuits Conference*, Feb 2012, pp. 426–428.

[18] J. Heichler, "An introduction to beegfs," November 2014. [Online]. Available: http://www.beegfs.de/docs/whitepapers/Introduction_to_BeeGFS_by_ThinkParQ.pdf

[19] P. Carns, W. L. III, R. Ross, and R. Thakur, "Pvfs: A parallel file system for linux clusters," in *4th Annual Linux Showcase and Conference*, 2000, pp. 391–430.

[20] N. Manchanda and K. Anand, "Non-uniform memory access (numa)." [Online]. Available: https://cs.nyu.edu/ lerner/spring10/projects/NUMA.pdf

[21] A. Rosen and K. Wang, "Steroid open flow service: Seamless network service delivery in software defined networks," in *First GENI Research and Educational Experiment Workshop*, 2012.

[22] "Cloudlab." [Online]. Available: https://cloudlab.us/

[23] T. Hwang, "Nsf geni cloud enabled architecture for distributed scientific computing," in *2017 IEEE Aerospace Conference*, March 2017, pp. 1–8.

[24] "Geni." [Online]. Available: http://www.geni.net/

[25] "Cloudlab hardware description." [Online]. Available: http://docs.cloudlab.us/hardware.html

[26] "Gnu wget." [Online]. Available: https://www.gnu.org/software/wget/