2010

Application-level optimization of end-to-end data transfer throughput

Esma Yildirim
Louisiana State University and Agricultural and Mechanical College

APPLICATION-LEVEL OPTIMIZATION OF END-TO-END DATA TRANSFER THROUGHPUT

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
Esma Yildirim
B.S., Fatih University, Istanbul, 2004
M.S., Marmara University, Istanbul, 2006
December, 2010

Acknowledgments

First and foremost I would like to offer my sincerest gratitude to my supervisor, Dr. Tevfik Kosar for his outstanding support and guidance throughout my studies and preparation of this dissertation. His patience and tolerance as well as his ability to see beyond the phenomenon helped me structure the foundations of this dissertation. One simply could not wish for a better or friendlier supervisor.

Second I would like to thank CCT (Center for Computation and Technology) for providing me a friendly working environment and every possible technology and technical support that I needed to develop my ideas.

Beyond everything else, nothing can replace the pure love and outstanding encouragement of my family which I felt in every moment I needed although they are thousands of miles away.

This project is in part sponsored by the National Science Foundation under award numbers CNS-0846052 (CAREER), CNS-0619843 (PetaShare), OCI-0926701 (Stork) and EPS-0701491 (CyberTools), and by the Board of Regents, State of Louisiana, under Contract Numbers DOE/LEQSF (2004-07), NSF/LEQSF (2007-10)-CyberRII-01, and LEQSF(2007-12)-ENH-PKSFI-PRS-03.

Table of Contents

Acknowledgments . . . . . **ii**

List of Tables . . . . . **v**

List of Figures . . . . . **vi**

Abstract . . . . . **ix**

Chapter 1: Introduction **1**
1.1 Contributions 4
1.2 Thesis Outline 5

Chapter 2: Background and Related Work **6**
2.1 Parallel Streams Optimization 6
2.2 Buffer Size Optimization 10
2.3 CPU and Disk Parallelism 14

Chapter 3: Analysis of End-system Bottlenecks **16**
3.1 Step 1: Effect of Parallel Streams on Disk-to-disk Transfers 16
3.2 Step 2: Effect of Parallel Streams on Memory-to-memory Transfers and CPU Utilization 17
3.3 Step 3: Effect of Striping and Removal of CPU Bottleneck 20
3.4 Step 4: Effect of Parallel Read& Write on Disk Throughput and Removal of Disk Bottleneck 20

Chapter 4: Models **25**
4.1 Parallel Stream Optimization Models 25
4.1.1 Modeling Packet Loss Rate 25
4.1.2 Increasing Curve Fitting with More Information 26
4.1.3 Logarithmic Modeling of Throughput Curve 28
4.1.4 Dynamic Extraction of Model Equation Order by Newton's Iteration 28
4.1.5 Full Second Order Model 30
4.1.6 Experimental Results 31
4.2 Balancing Buffersize vs Parallel Streams 39
4.2.1 Simulation Results and Discussion 40
4.2.2 LONI Experiments 42
4.3 Flow Model of End-to-end Throughput 46
4.3.1 Modeling CPU Utilization with Regression 48

Chapter 5: Optimization Algorithms **50**
5.1 Optimization Algorithm for Unknown Disk and Network Capacity in Homogeneous Resources(OPT_B) 50
5.2 Optimization Algorithm for Unknown Disk and Network Capacity in Heterogeneous Resources(OPT_H) 53

Chapter 6: Experimental Results **58**
6.1 Local Area Experiments for OPT_B Algoritm(LONI Network) 58
6.1.1 Disk-to-disk Transfers 59
6.1.2 Memory-to-memory Transfers 63
6.2 Wide-area Experiments for OPT_B Algoritm (LONI -TeraGrid Network) 65
6.2.1 Disk-to-disk Transfers 65
6.2.2 Memory-to-memory Transfers 68
6.3 Local Area Experiments for OPT_H Algoritm(LONI Network) 70
6.3.1 Disk-to-disk Transfers 71
6.3.2 Memory-to-memory Transfers 71
6.4 Wide-area Experiments for OPT_H Algoritm (LONI -TeraGrid Network) 74
6.4.1 Disk-to-disk Transfers 74
6.4.2 Memory-to-memory Transfers 76

Chapter 7: Conclusions **80**
7.1 Major Contributions 80
7.2 Future Work 81

References **82**

Vita **85**

List of Tables

3.1 CPU Utilization vs Parallel Streams 18

6.1 Testbed 58

List of Figures

1.1 Protocol bottleneck on LONI Linux Clusters 3

2.1 TCP Behavior 7

2.2 GridFTP results using tuned TCP buffers and parallel streams in LONI network . . . 13

3.1 Effect of parallel streams on GridFTP Disk-to-disk Throughput 17

3.2 Effect of parallel streams on GridFTP Memory-to-memory Throughput 17

3.3 Effect of Parallel Streams on CPU Utilization on Inter-node GridFTP Throughput . . 19

3.4 Effect of striping on GridFTP Throughput 20

3.5 Effect of Stripes on CPU Utilization on Inter-node GridFTP Throughput 21

3.6 Abe cluster (NCSA) MPI-IO disk throughput 23

3.7 Oliver cluster (LONI) MPI-IO disk throughput 24

3.8 Spider21 (CCT/LSU) MPI-IO disk throughput 24

4.1 The aggregate throughput results of GridFTP transfers of a 512MB file over 155ms latency wide area network 27

4.2 The prediction results of GridFTP transfers with randomly chosen stream levels . . . 32

4.3 Comparison of Prediction Models over 1Gbps link 37

4.4 Comparison of Prediction Models over 10Gbps link 37

4.5 Comparison of Prediction Models over 1Gbps-100 Mbps link 38

4.6 Comparison of Prediction Models over 100Mbps-1Gbps link 38

4.7 Comparison of Prediction Models over 100 Mbps local area link 39

4.8 Comparison of Prediction Models over 100 Mbps wide area link 39

4.9 Topology 40

4.10 Effect of parallel streams and buffer size under no cross traffic 40

4.11 Effect of parallel streams and buffer size with a cross traffic of 5 streams with 64KB buffer size 41

4.12 Effect of parallel streams and buffer size with a randomly generated internet cross traffic 41

4.13 Effect of parallel streams and buffer size over 10 Gbps LONI network 43

4.14 First technique for balancing buffer size and stream number 44

4.15 Second technique for balancing buffer size and stream number 45

4.16 Flow Model Graph 46

4.17 Regression model of CPU Utilization 49

5.1 Algorithm output(OPT_B) for Figure 6.3 53

5.2 Algorithm output(OPT_B) for Figure 6.1 54

5.3 Algorithm output(OPT_B) for Figure 6.12 54

5.4 Algorithm output(OPT_H) for Figure 6.21 57

6.1 Model Application on Disk-to-Disk Transfers between Eric and Oliver with 10G interfaces and 4 cores 60

6.2 Model Application on Disk-to-Disk Transfers between Eric and Poseidon head nodes with 10G interfaces and 8 cores 61

6.3 Model Application on Disk-to-Disk Transfers between Eric and Oliver with 1G interfaces 61

6.4 Transfer times for model application with 1GB sampling size vs data size 62

6.5 Model Application on Memory-to-memory Transfers between Eric and Oliver with 10G interfaces and 4 cores 63

6.6 Model Application on Memory-to-memory Transfers between Eric and Oliver with 1G interfaces 64

6.7 Transfer times for model application with 1GB and 2GB sampling size vs data size in memory-to-memory transfers 65

6.8 Model Application on Disk-to-disk Transfers between gridftp-abe and eric nodes with 10G interfaces and 4 cores 66

6.9 Model Application on Disk-to-disk Transfers between Lincoln and Poseidon clusters with 1G interfaces 67

6.10 Transfer times for model application with 2GB sampling size vs data size 67

6.11 Model Application on Memory-to-memory Transfers between gridftp-abe and Eric2 with 10G interfaces and 2 4-core nodes on source and 1 8-core node on destination . . 69

6.12 Model Application on Memory-to-memory Transfers between Lincoln and Eric with 1G interfaces 69

6.13 Transfer times for model application with 1GB and 2GB sampling size vs data size in memory-to-memory transfers between NCSA and LONI 70

6.14 Model Application (OPT_H) on Disk-to-disk Transfers between Oliver2 and Eric1&2 . 72

6.15 Model Application (OPT_H) on Memory-to-memory Transfers between Oliver2 and Eric1&2 73

6.16 Model Application (OPT_H) on Memory-to-memory Transfers between Poseidon(1G) and Eric1&2(10G) 73

6.17 Transfer times for model application(OPT_H) with 2GB sampling size vs data size in memory-to-memory transfers in LONI 74

6.18 Model Application (OPT_H) on Disk-to-disk Transfers between Lincoln(1G) and Eric(10G) 75

6.19 Model Application (OPT_H) on Disk-to-disk Transfers between Abe(10G) and Eric2(10G) 76

6.20 Transfer times for model application(OPT_H) with 2GB sampling size vs data size in disk-to-disk transfers between NCSA and LONI 77

6.21 Model Application (OPT_H) on Memory-to-memory Transfers between Lincoln(1G) and Eric1&2(10G) 78

6.22 Model Application (OPT_H) on Memory-to-memory Transfers between Abe(10G) and Eric2(10G) 78

6.23 Transfer times for model application(OPT_H) with 2GB sampling size vs data size in memory-to-memory transfers between NCSA and LONI 79

Abstract

For large-scale distributed applications, effective use of available network throughput and optimization of data transfer speed is crucial for end-to-end application performance. Today, many regional and national optical networking initiatives such as LONI, ESnet and Teragrid provide high speed network connectivity to their users. However, majority of the users fail to obtain even a fraction of the theoretical speeds promised by these networks due to issues such as sub-optimal protocol tuning, disk bottleneck on the sending and/or receiving ends, and processor limitations. This implies that having high speed networks in place is important but not sufficient for the improvement of end-to-end data transfer throughput. Being able to effectively use these high speed networks is becoming more and more important.

Optimization of the underlying protocol parameters at the application layer (i.e. opening multiple parallel TCP streams, tuning the TCP buffer size and I/O block size) is one way of improving the network transfer throughput. On the other hand, end-to-end data transfer throughput bottleneck on high performance networking systems occur mostly at the participating storage systems rather than the network. The performance of a storage system heavily depends on the speed of its disk and CPU subsystems. Thus, it is critical to estimate the storage system's bandwidth at both endpoints in addition to the network bandwidth. Disk bottleneck can be eliminated by the use of multiple disks (data striping), and CPU bottleneck can be eliminated by the use of multiple processors (parallelism).

In this dissertation, we develop application-level models to predict the best combination of protocol parameters for optimal network performance, including the number of parallel data streams, protocol buffer size; and integration of disk and CPU speed parameters into the performance model to predict the optimal number of disk and CPU striping for the best end-to-end data throughput. These models will be made available to the community for use in data transfer tools, schedulers, and high-level planners.

Chapter 1
Introduction

The advancement of science in many fields (e.g. coastal and environmental modeling, bioinformatics, medical imaging, fluid dynamics and high energy physics) with their emerging data management needs is only possible with end-to-end network capabilities that will support their Petascale computational and data requirements. With the recent developments in the optical network technology, which is exceeding 100Gbps data transport capacity, the utilization of network resources has brought a challenge to the current middleware solutions to provide distributed Petascale science [31].

The high-speed networks have constructed a base for the Petascale generation of supercomputers and it has become a must for the host-systems, the protocol stack and the network developers to be consistent with each other to be able to provide high-speed I/O capabilities. In the current techniques, achieving multiple Gbps throughput conventionally over TCP-based networks has become a burden. The throughput rate achieved by today's IP networks is limited by the performance of TCP which is a reliable data transport protocol and the basis of default application-level transfer protocols (e.g. FTP, GridFTP). Over the years, many variants of TCP have been implemented to improve its performance, however it still remains as the most-widely adopted data transport protocol.

Apart from the low-level transport protocols implemented, there has been a number of high-level methods to optimize the throughput of data transfers regardless of the underlying protocol used. Tuning underlying end-system parameters and opening parallel streams is a way of doing it and widely used in many application areas from data-intensive scientific computing to live multimedia and peer-to-peer paradigms.

It is shown that parallel streams achieve high throughput by mimicking the behavior of individual streams and get an unfair share of the available bandwidth [26, 19, 4, 11, 8, 16, 20]. On the other hand, using too many simultaneous connections reaches the network on a congestion point and after that threshold, the achievable throughput starts to drop down for low-speed networks. Unfortunately it is difficult to predict the point of congestion and since it is variable over some parameters which

are unique in both time and domain. The prediction of the optimal number of streams is hence very challenging without some parameters of current network conditions such as available bandwidth, RTT, packet loss rate, bottleneck link capacity and data size.

It is easier to optimize the network parameters in the case of bulk data transfers. After some point in time, enough historical information is gathered and the parameters are optimized according to the condition of the network. This type of optimization has already been used with Stork data scheduler [18] before. However, for individual data transfers, the optimization becomes harder. Instead of relying on historical information, the transfer should be optimized based on instant feedback. This optimization could be done by achieving optimal number of parallel streams to get the highest throughput. However, an optimization technique not relying on historical data in this case must not cause too much overhead due to gathering instant data which would be larger than the speed up gained with multiple streams for a particular data size. Gathering instant information for prediction models could be done by using network performance measurement tools. However it is very difficult to choose a measurement tool for efficient and accurate information collection. Although difficult, the prediction of the optimal parallelism level for a data transfer regardless of the underlying network characteristics is not impossible(whether it is a high-speed network or a low-speed WAN or LAN).

With the development of recent high-bandwidth networking technologies reaching 100Gbps speeds, the bottlenecks over the end-systems have become the major factor that affects the upper limit of the end-to-end data transfer speed. End-systems that invoke and receive data transfers may range from supercomputers to computational clusters or to hosts of varying capabilities [31]. They have potential bottlenecks such as disk throughput, CPU speed, MTU and window size. However these bottlenecks could be overcome by proper tuning of the systems based on end system properties and architecture.

Network is the major source of bottleneck especially for low-bandwidth architectures. The systems connected with this kind of network, usually consist of single CPU/node, single disk access and NICs under or equal to 100Mbps. The possible causes of low data transfer performance can be listed as *latency*, *current network traffic*, *protocols used* and *untuned buffer size*. While the first two problems

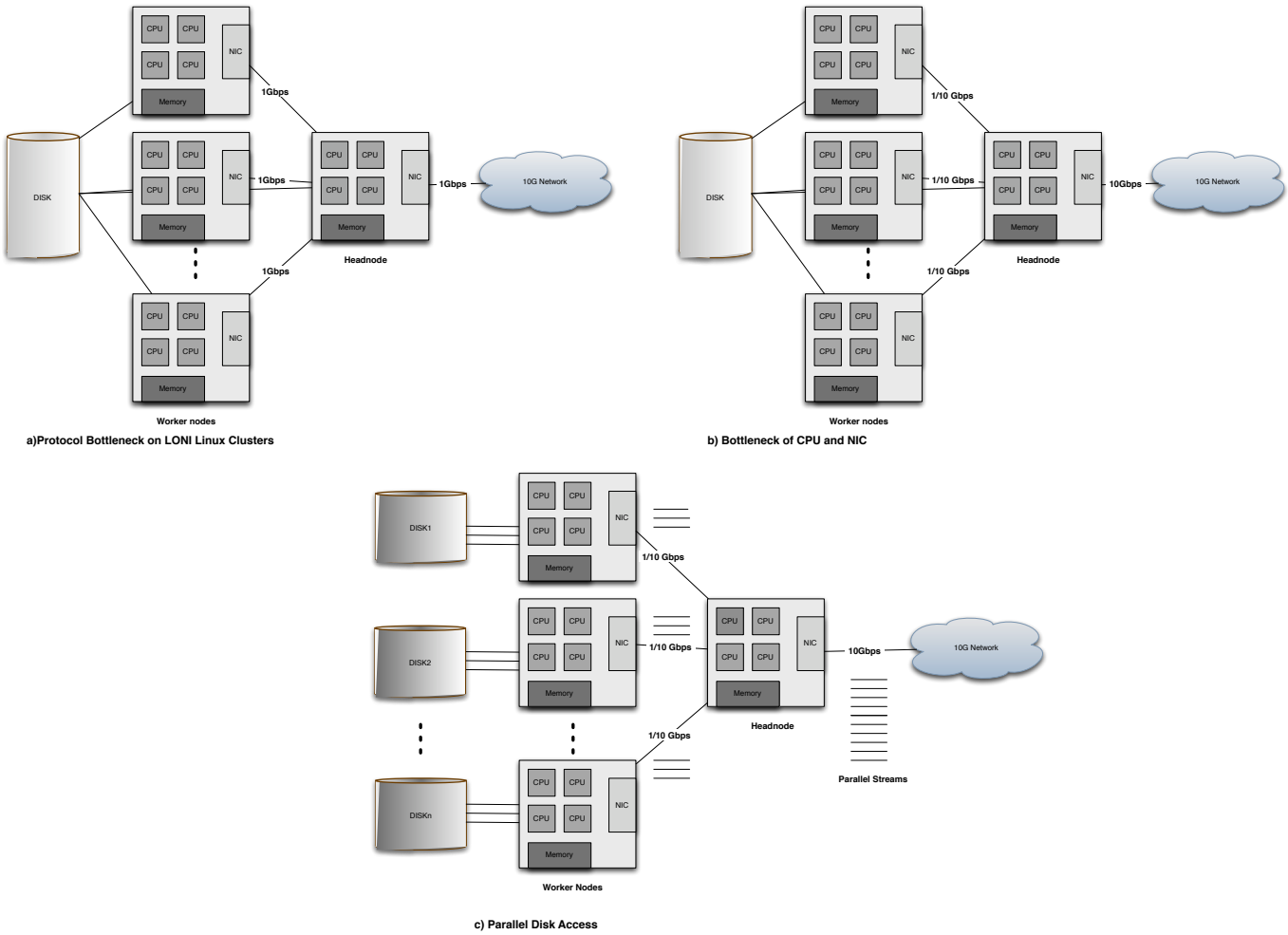


FIGURE 1.1: Protocol bottleneck on LONI Linux Clusters

are directly related to the network bottleneck, the latter two are related indirectly since these settings are configured at the end-system.

The end-systems that are connected with high-speed networks, usually consist of parallel clusters, super-computers or parallel storage systems. The first and foremost problem on those end-systems is again the protocol used. For example, TCP protocol is not designed for those type of networks and can not utilize the most of the available bandwidth. Usage of multiple parallel streams and correct buffer size could solve the problem in the application level. Figure 1.1.a presents an example architecture on the LONI network. In this architecture, the head node and the worker nodes have 1Gbps NICs. The maximum buffer size is set to 128KB by default. Without any optimization, a throughput of around 100Mbps is achieved. However with proper use of parallel streams, it increases to around 900Mbps.

Usage of parallel streams or large buffer sizes places a burden on CPU as well as NIC. For parallel architectures, use of multiple CPUs and NICs may address the problem. In a typical distributed shared memory architecture, while the speed of the NIC on the head node is compatible to the speed of the network, the ones on the worker nodes may or may not. Considering that single disk access is available, the possible causes of bottleneck that may occur in such a system could be listed as *CPU speed*, *protocol settings* and *NIC on worker nodes*. Figure 1.1.b shows an example architecture in which the head node has a NIC of 10Gbps whereas the worker nodes have 1Gbps or 10Gbps NICs. For the first case (1Gbps NIC on the worker node), two problems may occur: the CPU on the headnode may not compete to the speed of NIC or if it is a worker node the NIC itself becomes the bottleneck. For the second case, the only possible cause of bottleneck is the CPU considering memory-to-memory transfers where disk is not part of the end-to-end data transfer path.

Many parallel systems that are connected via high-speed networks have parallel storage systems. Disk access rate is a major source of bottleneck in single disk access systems. However in parallel storage systems, this bottleneck could be overcome by reading/writing data in stripes. While overcoming this bottleneck, we could generate new bottlenecks since other parts of the system(e.g. CPU, NIC) may not keep up to it. In this case the possible causes of bottleneck could be listed as *disk speed*, *CPU speed*, *protocol settings* and *NIC*. Figure 1.1.c presents an example architecture.

In this dissertation, we propose an application-level approach to overcome protocol and end-system bottlenecks and develop models to provide the optimum end-to-end data transfer throughput. These models decide on the optimal settings(e.g.number of parallel streams per stripe, buffer size, number of stripes per node, number of nodes) for a specific data transfer considering the current architecture of the systems and make projection of the available amount of bandwidth for the users.

1.1 Contributions

We have developed models that address the dynamic orchestration of multi-domain network resources to provide end-to-end throughput optimization. From a broader point of view, the collaborative tuning of end-system resources (e.g. storage, CPU) in conjunction with the network resources is realized.

The developments of high-speed networks that connect supercomputers and other parallel systems as well as the lack of a transport-level protocol that will utilize the resources, led us to the dynamic application-level optimization of these resources. The models presented provide the users with the optimal parameters to obtain the highest end-to-end throughput by using existing protocols and tools. The major contributions of this work can be listed as follows:

- We provide optimization for data transfer throughput in the application-level without any need to change transport protocols.
- We develop models by using as little information as possible at the same time providing accuracy and scalability regardless of the architecture.
- We enable utilization of high-speed networks by removing the bottlenecks in the end systems and achieve this through optimal modeling of CPU parallelism and Disk striping in today's sophisticated architectures such as supercomputers, parallel disk systems and clusters.
- We provide the user with optimization parameters specific to their transfers.

1.2 Thesis Outline

The chapters of the dissertation are organized as follows: In Chapter 2, we present the background information and the previous work related to parallel streams, buffer size and CPU and disk parallelism. Chapter 3 analyzes the possible causes of bottlenecks that could occur in end-systems and discusses how to overcome these bottlenecks. In Chapter 4, we present novel models about parallel streams and buffer size optimization and a thorough flow model which includes the removal of end-system bottlenecks. In Chapter 5, we present algorithms that apply the flow model to improve the end-to-end data transfer throughput. Chapter 6 has detailed experimental results which validate the models presented in this dissertation. In Chapter 7, we conclude this dissertation and in Chapter 8, we present the list of publications about this work.

Chapter 2
Background and Related Work

The existing high-speed networks suffer from the inadequacy of the current protocols in use which were not designed taking into account their properties. We focus on TCP since it is the most widely adapted protocol and one of the most important reasons of bottleneck in high-speed networks because of its design goals.

TCP is a transport layer protocol designed to utilize the network bandwidth giving importance to fairness among the flows sharing the network. It has two phases that are named "Slow start" and "Congestion Avoidance" (Figure 2.1). The congestion window size is the number of packets the sender is allowed to send. The higher the congestion window the higher the throughput. In the slow start phase, the congestion window starts from 1 packet and is exponentially increased to utilize the throughput quickly until a packet loss event occurs. Then the congestion window is divided by half and starts to increase linearly. This is known as the *additive increase - multiplicative decrease* property(AIMD) [29]. When a timeout occurs the loop turns to the beginning congestion window size and enters the slow start phase again. This property of TCP ensures fairness, however it gives poor performance in terms of throughput. Therefore other methods have been developed to compensate its poor performance.

There has been a number of studies that try to optimize the network and the end-systems to obtain the maximal end-to-end throughput in terms of several parameters such as the parallel stream number, window size, MTU(Maximum Transmission Unit) size, load sharing with IRQ(Interrupt ReQuest) bonding and interrupt coalescing. Most of the optimizations are done manually and dynamic optimization methods are scarce.

2.1 Parallel Streams Optimization

The studies that try to find the optimal number of streams are so few and mostly based on approximate theoretical models [10, 21, 3, 17]. They all have specific constraints and assumptions. Also the correctness of the proposed models are mostly validated with simulation results only.

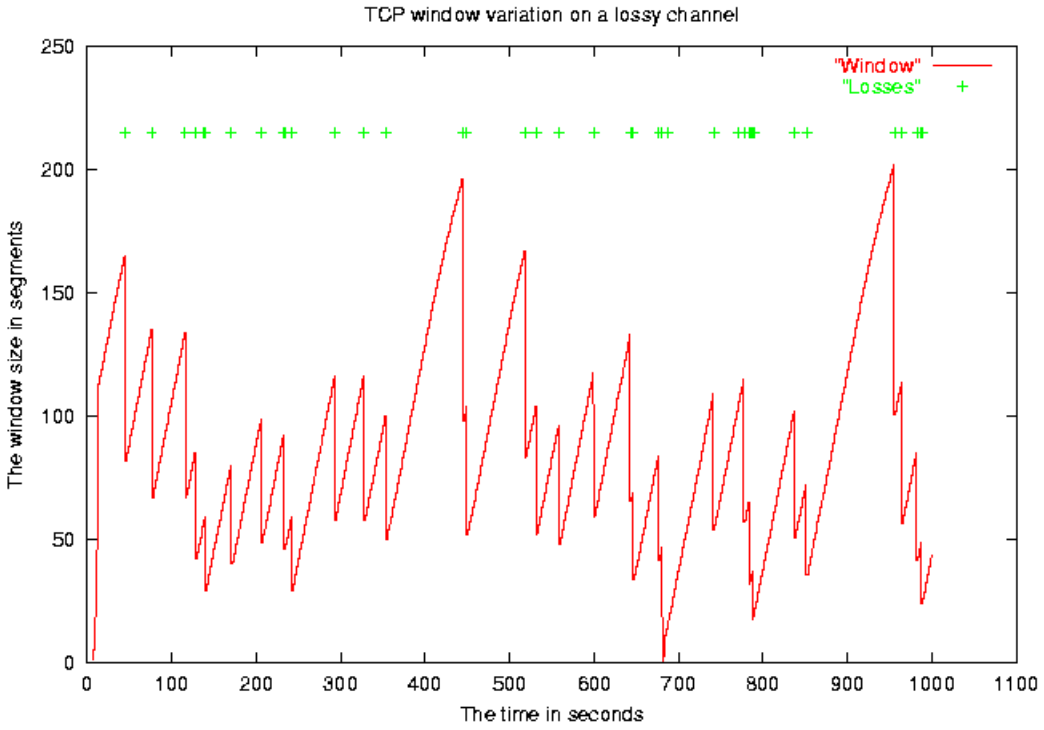


FIGURE 2.1: TCP Behavior

Hacker et al. claim that the total number of streams behaves like one giant stream that transfers in capacity of total of each streams' achievable throughput [10]. In this model, the achievable throughput depends on three parameters: Round trip time, Maximum Segment Size and Packet Loss rate. The following equation presents an upper bound on the achievable throughput of n streams :

$$Th <= n\frac{MSS}{RTT}\frac{c}{\sqrt{p}} \tag{2.1}$$

RTT represents *round trip time*, MSS represents the *maximum segment size*, p represents the *packet loss rate* and c is a constant.

However, this model only works for uncongested networks and assumes the packet loss rate is stable and same for each connection and does not increase as the number of streams increases. At the point the network gets congested, the packet loss rate starts to increase dramatically and the achievable throughput starts to decrease. So it is important to find that point of knee in packet loss rate. Overall

this model requires too much information that is also hard to gather and it does not give an answer to at what point the network will be congested.

Dinda et al. models the bandwidth of multiple streams as a partial second order equation and needs throughput measurement of two different stream numbers to predict the others [21]. The model tries to solve Equation 2.1 by computing the relationship between p, RTT, and n. MSS and c are considered as constants. The relationship is represented by a new variable p'_n which is equalized to a partial second order polynomial which is believed to be best suited [21]:

$$p'_n = p_n \frac{RTT_n^2}{c^2 MSS^2} = a'n^2 + b' \tag{2.2}$$

After placing p'_n in Equation 2.1, total throughput of n streams is calculated as follows:

$$Th_n = \frac{n}{\sqrt{p'_n}} = \frac{n}{\sqrt{a'n^2 + b'}} \tag{2.3}$$

a' and b' are parameters to be found by measurements. To be able to solve this equation, two achievable throughput measurements for two different parallelism levels are needed. The only possible way to find those throughput values is either use a tool that has the capability to do parallel transfers or to use information about past transfers. The experimental results claim that the parallelism level correctly can be predicted, however in those results the aggregated throughput of connections increases and then becomes stable but never falls down. Hence it does not take into account that opening too many streams can place a burden over the network and the end-system and causes the throughput to decrease after the optimal point.

In another model, the total throughput always shows the same characteristics [3] depending on the capacity of the connection as the number of streams increases and 3 streams are sufficient to get a 90% utilization. This model bases its correctness on the fact that opening too many connections imposes a processing overhead and leaves smaller bandwidth to other flows. It is noted that for single stream situation, a TCP connection may utilize only 75% of the bandwidth because of its AIMD (additive increase and multiplicative decrease) property. Only with parallel streams we could increase this ratio. Because only a small subset of the connections undergoes multiplicative decrease during congestion, this helps parallel streams to recover quickly. Assuming only one of the connections

undergoes a multiplicative decrease at any time, the following formula can be used to predict the throughput [3]:

$$Th_n = C\left(1 - \frac{1}{1 + \frac{1+B}{1-B}n}\right) \tag{2.4}$$

B is 1/2 for TCP connections as it decreases its window by half for the multiplicative decrease and C is the capacity of the link. There are two issues that need to be solved for that approach to be applied. First of all, we need to know the bottleneck link capacity for the overall connection to be able to produce correct results since this formula gives the throughput over a single link. Second, this formula gives the total number of streams that survive to get this aggregate throughput however if we need to know the additional number of streams to open then we must have an idea about how many other streams are using the link as cross traffic. We must find a way to determine the existing flow number.

A new protocol study [17] that adjusts sending rate according to calculated backlog presents a model to predict the current number of flows which could be useful to predict the future number of flows. A target backlog is computed from the current measured bottleneck to achieve the stated goals. Four input parameters are needed for calculations: Capacity of the bottleneck link, average and minimum round trip times and packet loss rate. With these inputs it is possible to derive the cross-traffic throughput and average number of flows sharing the bottleneck link. If we know the average number of flows sharing the bottleneck link we can combine this with the model in [3] . Say that we want a 98% link utilization and we know the capacity of the bottleneck link. We can calculate the optimal n and subtract the average number of flows sharing the link. So we find the number of additional streams to open.

$$n_{add} = n_{opt} - n \tag{2.5}$$

However the information for those calculations can be gained in the low level protocol layer. For example the sending rate can only be decided by the underlying protocol which depends on setting the window size dynamically.

2.2 Buffer Size Optimization

Another important parameter to be tuned for high throughput is the TCP buffer size. Buffer size parameter affects maximum number of packets that will be on the fly before the sender will wait for an acknowledgement. If a network buffer is undersized then the network cannot be fully utilized. However, if it is oversized there could also be degradation in the throughput because of packet losses hence window reductions. Usually it is tuned manually by the application users or in the kernel level of the operating system.

A common method to tune the buffer size is to set it to the twice the value of $bandwidth \times delay$ product (BDP). However this assumption of making the buffer size as large as the twice of the BDP only holds when there is no cross traffic along the path which is quite impossible. So it brings the question of whether to use the capacity of the network or the available bandwidth as well as minimum RTT or maximum RTT. Hence there is a quite variety in the understandings of the bandwidth and delay concepts. Below is a list of different meanings of BDP [15]:

- BDP1: $B = C \times RTT_{max}$
- BDP2: $B = C \times RTT_{min}$
- BDP3: $B = A \times RTT_{max}$
- BDP4: $B = A \times RTT_{min}$
- BDP5: $B = BTC \times RTT_{ave}$
- BDP6: $B = B_{\infty}$

In the above equations, B represents the buffer size, C represents the capacity of the link, A represents the available bandwidth and RTT is the round trip time. BTC in BDP5 is the average throughput of a bulk congestion-limited transfer and calculated based on the congestion window size. Finally B_{∞} in BDP6 is a large value which is always greater that the congestion window so that the connection will always be congestion-limited.

Most of the tuning methods described in the literature do the tuning in the kernel level and approaches that use application level auto-tuning are so few [15], [24]. The methods that use one of

the above equations usually rely on tools to take the measurement of available bandwidth and RTT and do not consider the effect of the cross traffic and congestion created by using large buffer sizes.

The techniques that need modifications to the kernel [6], [25], [30], [34] is usually based on dynamic changes to the buffer size during the transfer based on the congestion window or flow control window parameters. The approach presented in [6] requires changes to the Kernel Stack. Based on the current congestion window, RTT and server read time, they calculate the following congestion and set the buffer variables based on the current and next congestion window sizes. Those variables determine the upper bound of the socket buffer before it can accept data from application as well as the lower bound of free space and current number of acknowledged bytes. In that sense this approach is similar to BDP5. Another study [25] is also similar to the previous where the sender buffer size is adjusted based on the congestion window. Also the buffer memory is fairly shared among the connections. The receive buffer is sufficiently large so that it will not limit the throughput. Other two of the competitive techniques which are widely used are Dynamic Right-Sizing [34] and Linux 2.4 Auto-Tuning [30]. DRS is basically a receiver -based approach where the receiver tries to estimate the $bandwidth \times delay$ product by using TCP packet header information and time stamps. Instead of using static flow control windows, the advertised receive window is dynamically changed so that the sender is not limited by the flow control. On the other side, Linux auto-tuning is a memory management technique in which it increases or decreases window size continuously based on the available memory and socket buffer space. The preferable technique to be used may depend on the data transfer characteristics. For example, while Linux 2.4 auto-tuning is good for large number of small connections, Dynamic Right Sizing is preferred for smaller number of large connections such as bulk data transfers or FTP [35]. Also the latency is another big factor for the selection of window size tuning technique. While tuning window size does not have a big effect for a small latency transfer, it has a big effect for high latency transfers.

There are also techniques that are applied in the application level without changing the kernel and the protocol [15], [24], [12], [22]. Those methods are usually static and once the buffer size is set it does not change throughout the transfer. In [12] the estimated throughput of a connection is calculated based on packet loss probability, RTT and Retransmission time out based on the model

presented in [23]. Then by using the BDP product the buffer size is determined: $Buffersize = EstimatedThroughput \times RTT$. In [22], a series of ICMP messages are sent out and the packet length is divided by the time difference between two consecutive packets and by dividing them to the RTT, the available bandwidth of the bottleneck link is calculated. Then by multiplying it with the RTT the buffer size is determined. The study in [15] and [24] separates the buffer of a congested path from a non-congested path. The approach they propose finds the efficient buffer size for especially non-congested paths. In a non-congested path, the optimal buffer size is found by considering the existing cross traffic. They propose an application layer tool called SOBAS, where in uncongested paths it limits the buffer size so it does not overflow the buffers of the bottleneck link and in congested paths it does not limit the transfer window so that the buffer size increases and becomes congestion limited. They send periodic UDP probing packets and calculate the throughput based on those probes. When the throughput becomes stable for some period of time they limit the receiver buffer according to that value.

Although the buffer size parameter is properly tuned, it does not show a better performance than using parallel streams because parallel streams recover from packet loss quickly rather than a buffer tuned single stream. Using a tuned buffer size for a single stream provides a good performance comparing to an untuned stream. Using parallel streams provides an even better performance although the buffer is untuned. A good combination of tuned buffer size with parallel streams can even improve the performance further [7]. With a tuned buffersize however, less number of streams are needed compared to untuned parallel streams(Figure 2.2). If a good balance among the buffer size and parallel streams could be developed, we could find an optimal combination in which excess number of streams are avoided but at the same time better performance is gained.

Although a decent number of studies have been presented in buffer size tuning, there are only a few studies that tries to derive a mathematical model to find the relationship between buffer size and number of parallel streams. The study in [14] models the throughput of parallel streams as multiple continuous time models of TCP congestion control mechanism. When the window size is greater than the BDP($bandwidth \times delay$) product, they model the relation among number of streams, throughput and bottleneck link bandwidth as follows:

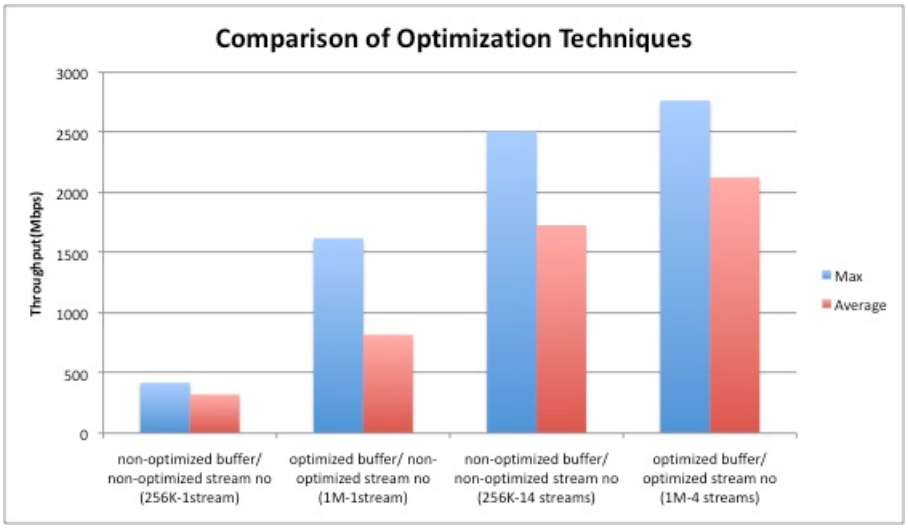


FIGURE 2.2: GridFTP results using tuned TCP buffers and parallel streams in LONI network

$$B = NT \simeq \frac{N}{2R}(-3 + \frac{\sqrt{6+21p}}{\sqrt{p}}) \tag{2.6}$$

where B is the bottleneck link bandwidth, N is the number of streams, T is the throughput of a single stream in steady state and p is the packet loss probability. When the window size is not properly tuned throughput is defined as W/R where W is the buffer size and R is the round trip time. Packet loss is considered as negligible and the derivations lead to the following equation:

$$N = (3BR - 3W - \sqrt{3}\sqrt{9B^2R^2 - 16BRW + 7W^2}) \times \frac{BR}{9BR - 6W} \tag{2.7}$$

Another study represents the relationship between number of parallel streams, buffer size and round trip time by a single regression equation [5]. However this equation again considers a no-loss network. In Equation 2.8, the BDP is equalized to the proportion of the optimal buffer size (B_o) and optimal parallel stream number(P_o). C_2 is added to the equation because the throughput may not be exactly proportional to the buffer size. C_1 is also added to represent the proportional relation. However those constants are derived from averaged results of experiments and may be different on different types of networks.

$$B \times R = C_1 \times P_o \times {B_o}^{C_2} \tag{2.8}$$

The existing models are scarce and derived for no-loss networks and they consider the cases where the buffer size is not properly tuned but they do not consider the case where the buffer is also properly tuned and also using the parallel streams for tuned buffers as in [7]. There is no result comparing a less tuned buffer size with using more streams or a properly tuned buffer and using less number of streams.

2.3 CPU and Disk Parallelism

The development of high speed networks caused the end-systems to be the bottleneck for high data throughput rather than the network. The major bottlenecks include the disk and the CPU limitations. Some of the optimizations to overcome the disk bottleneck are tuning I/O block size, tuning I/O scheduler and prefetching [28]. I/O block size defines the number of bytes read/written at a time from/to disk. I/O schedulers may perform different based on the load of the system and the characteristics of the transfers. While some schedulers work better than the others in busy machines, they can perform worse in idle machines. Also they differ in CPU usages as well. Prefetching can improve disk speed by reading ahead for large data sizes however it can increase access times for small data sizes. Even if all the parameters are tuned optimally, still the available high-bandwidth network may not be utilized fully and limited by the performance of single disk. In this case, an optimal level of striping the transfer would be a way to overcome this bottleneck. However, the cost of that approach must be carefully calculated and models must be devised to decide on the best level of striping.

The second major bottleneck in utilizing high-speed networks is the CPU. One of the techniques that is used to overcome this bottleneck is interrupt coalescing. When there is a high interrupt rate usually on the headnode of a supercomputer or cluster architecture only one interrupt is generated for multiple packets. Another technique is called IRQ bonding [28] where specific interrupts are shared among the CPUs. Again it is really important to decide on the optimal level of parallelism about the CPU usage.

The GridFTP Striped Server [2] provides an architecture, that supports striping and partial file transfer. According to this architecture data could be striped or interleaved across multiple servers as in a parallel file system or DPSS [32] disk cache. Data sources and sinks may come in different

forms such as clusters with local disks, clusters with parallel file systems, archival storage systems and geographically distributed data sources. A common configuration could be nodes connected by 1 Gbps connections to a switch that is connected to the external node with 10Gbps or faster. Their experimental results showed that they could achieve a 27.3 Gbps memory-to-memory transfer and 17 Gbps disk-to-disk transfers in a 30 Gbps network. Striped transfers are performed in combination with parallel transfers. In numerous experiments, increased streams did not equate to increased performance, only as they approached the bottleneck link speed the number of streams begin to have an effect. There is a knee point when the streams begin to compete with themselves or overflow router buffers. Also disk-to-disk transfers heavily depend on the read and write speed of the parallel file systems they used and limited by them.

Dmover [27] is used to schedule bulk data transfers between GridFTP nodes. Dmover jobs contain the locations and paths of the files to be transferred. If a stripe count is specified, file transfers will occur in a striped mode. Both architectures provide tools to do parallelism and striping however do not provide the optimal level of parallelism. Dynamic decision of this number needs insight to many factors which in the following section we analyze the effects of those factors in detail.

Chapter 3
Analysis of End-system Bottlenecks

Using parallel streams is a common method to remove the inadequacy of TCP protocol for long-fat networks. They provide multiple amounts of the throughput achievable by a single stream at the expense of losing fairness among other flows using the network. For uncongested networks, this issue is not a problem which is often the case for high-speed networks because they usually have unused bandwidth due to improper use of end-systems. However using excessive number of parallel streams is also not a good way to utilize the resources. There are many reasons that prohibit the usage of excessive parallel streams. One reason is that it could reach the network to a congestion point where packet loss rate increases and causes a drop-down in throughput. Another reason is that it could reach the end-system to its limits such as the CPU load, disk access speed and available NIC capacity. Another method that is especially used to overcome the disk bottleneck is striping which means accessing different parts of data on multiple disks in parallel.

In this chapter, we present a step-by-step analysis of end-system bottlenecks and the effect of using parallel streams and stripes on the end-to-end throughput by using GridFTP and MPI-IO. Three different networks are used in the experiments conducted. The first network is a state-wide high-speed network of Louisiana which is called LONI [36]. It has a 10Gbps backbone and connects 128 node Linux clusters with 4-8 core processors on each node. The latency of paths between clusters is short (e.g. up to 8msecs). The second network is the Teragrid [37] which is a nation-wide high-bandwidth network that has a 30 Gbps backbone. In the third setting, we use the inter-node connection which means node-to-node transfers in the same cluster to remove the effects of the network.

3.1 Step 1: Effect of Parallel Streams on Disk-to-disk Transfers

The throughput that is gained by the users while doing a data transfer over the high-speed networks without doing any optimization is usually the disk throughput. In this section we present the throughput of non-optimized disk-to-disk transfers and see the effect of using parallel streams. In

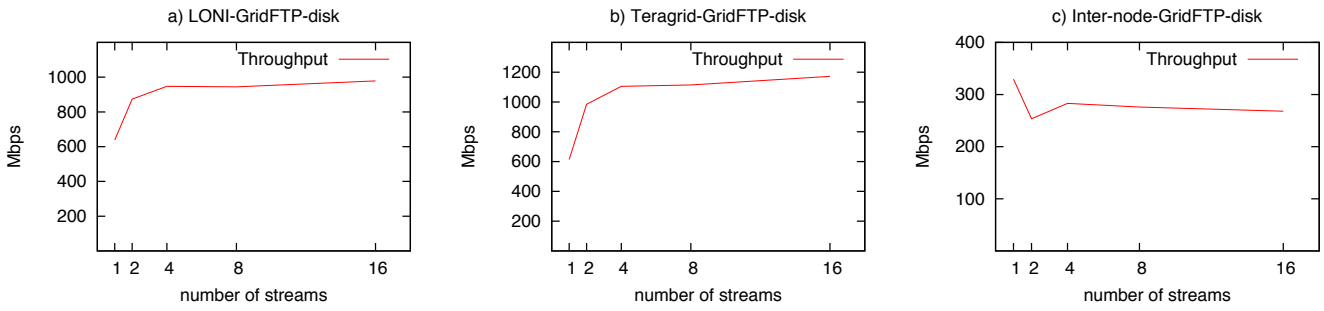


FIGURE 3.1: Effect of parallel streams on GridFTP Disk-to-disk Throughput

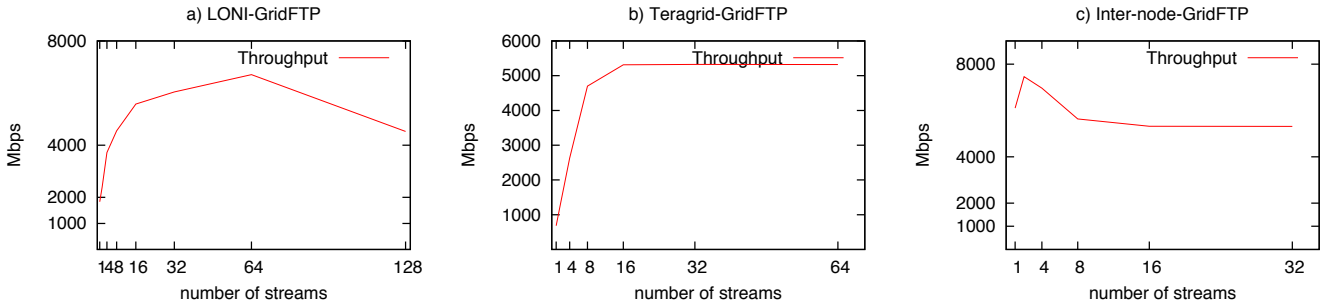


FIGURE 3.2: Effect of parallel streams on GridFTP Memory-to-memory Throughput

Figure 3.1, we present the disk-to-disk throughput of three different environments mentioned in the upper section. The disk-to-disk transfers for the LONI and the TeraGrid networks are conducted over the Lustre parallel file system [13] stored on each cluster, while for the inter-node transfers, the local hard drive and file system is used. The throughput for the LONI and Teragrid transfers are around 600Mbps for non-optimized single stream transfers and it increases when we use parallel streams to the range of 900-1100Mbps. On the other side, the inter-node transfers show that the throughput even falls down to 250Mbps from 350Mbps if we use parallel streams. Considering the network and the network interfaces support 10Gbps, these results show how poor the throughput is for the users of these networks and clusters.

3.2 Step 2: Effect of Parallel Streams on Memory-to-memory Transfers and CPU Utilization

In this section, we eliminate the disk from the end-to-end data transfer path to see the effect of the disk bottleneck. Figure 3.2 presents the average memory-to-memory throughput of parallel streams with the same settings used in the previous section. In general, the throughput increases as the

TABLE 3.1: CPU Utilization vs Parallel Streams

Average Receiver CPU Utilization(%)							
Streams	1	2	4	8	16	32	64
Oliver-Poseidon(LONI)	8,410	4,668	28,643	17,655	29,194	29,554	37,795
Spider-Abe(TeraGrid)	2,140	7,781	13,897	25,178	32,692	34,774	32,841

stream number increases, after reaching its peak point, it either continues stable or starts to decrease either due to network or end-system bottlenecks. In each setting the nodes used have 10Gbps network interface, henceforth we remove the possibility of NIC bottleneck. The highest throughput is gained with inter-node transfers which is around 7.5 Gbps. Considering the network is eliminated and the interconnects among the nodes do not present a bottleneck, this throughput is still low for 10Gbps network interfaces. We look into the CPU utilization results to find out why using parallel streams is not sufficient.

In Figure 3.3, we present the CPU utilization of both sender and receiver side of Inter-node transfers. Two I/O nodes (Spider 21-22) of a cluster in LONI network are used. These nodes have 8 Intel processor cores. One important observation is that the utilization is higher in the receiver side (Spider22); hence it plays the ultimate role to set the achievable throughput limit. The sender (Spider22) starts with a 50% cpu utilization on single core and goes up to over 80% as the number of streams increases. The receiver utilization on the other hand is around 100% even for 1 stream and it pushes the limits of a single CPU as the stream number goes up. But it never uses more than two processors and a total of %140 percent CPU out of two cores. Furthermore the utilization of two cores is not even. Yet the throughput starts to decrease after 2 streams. In average, the receiver utilization is around 15% for a total of 8 cores. Also utilization is correlated with the throughput achieved. Similar results were gained for LONI and TeraGrid testbeds. Table 3.1 presents the average utilization values for the receiver side. The values are around 35% at maximum. However these nodes have only 4 cores and in this case this value is equal to 17.5% comparing with the previous setting of Spider. Since the LONI tests were conducted on busy nodes and only the transfer utilization is parsed out, the results for that are more unstable, however it still shows correlation with the throughput achieved in Figure 3.2. We have also seen that only single core utilization was maxed out.

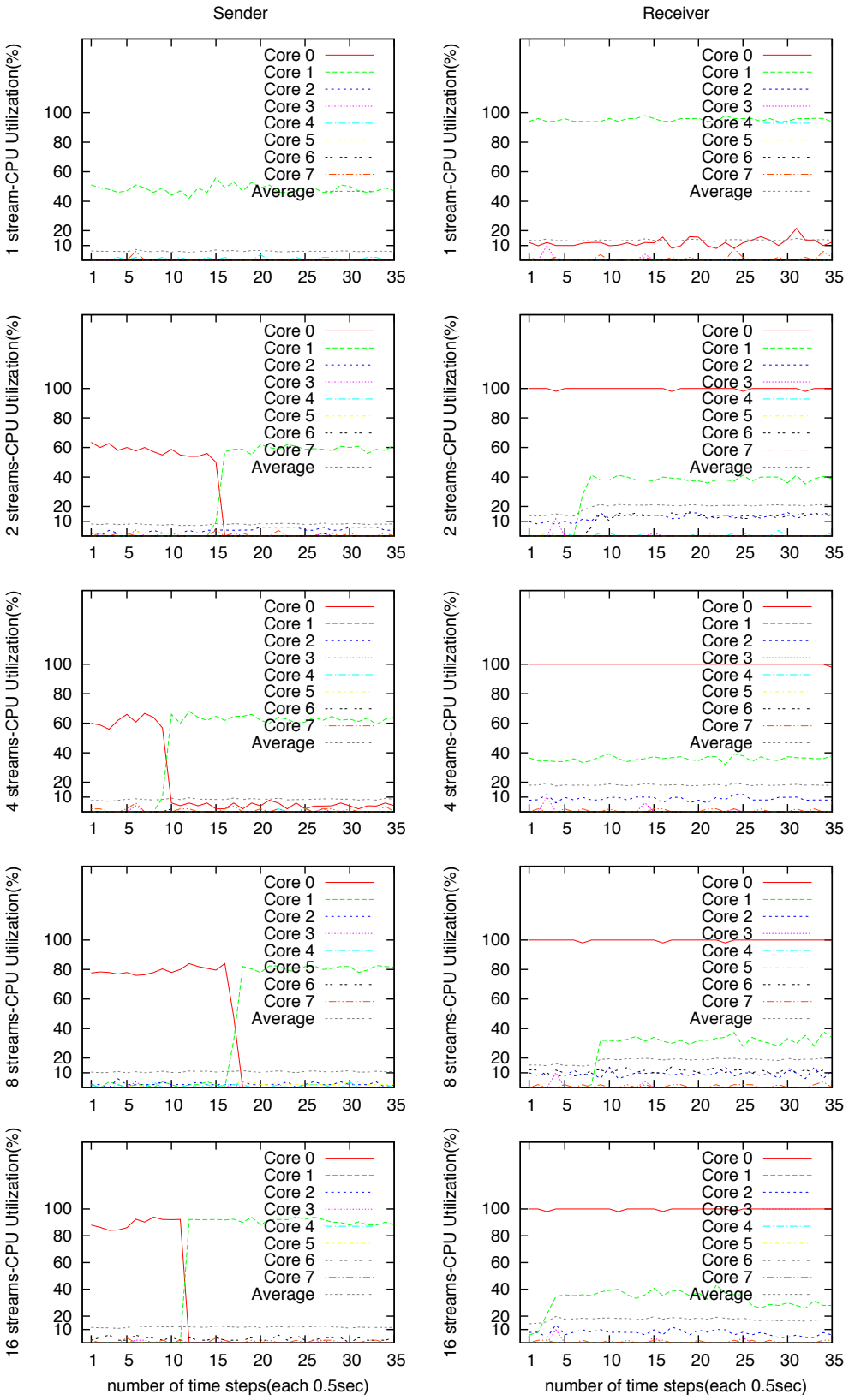


FIGURE 3.3: Effect of Parallel Streams on CPU Utilization on Inter-node GridFTP Throughput

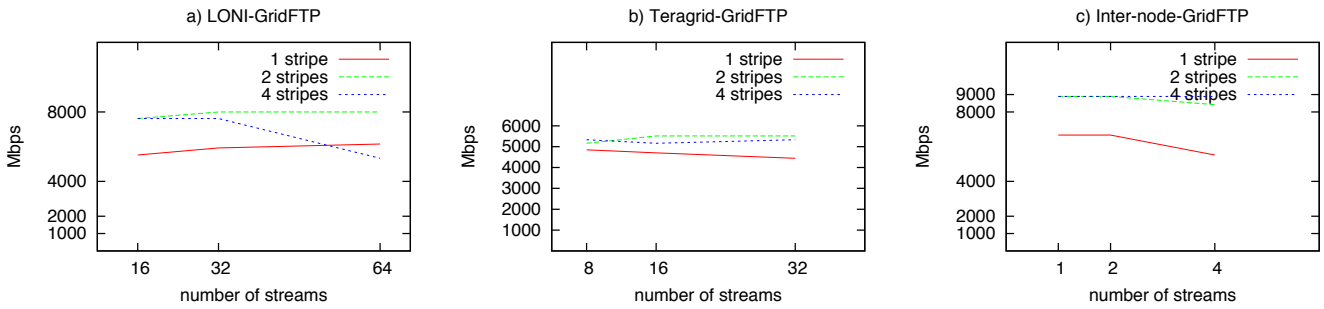


FIGURE 3.4: Effect of striping on GridFTP Throughput

3.3 Step 3: Effect of Striping and Removal of CPU Bottleneck

To understand the source of the bottleneck, a second set of transfers are performed by adding stripes and using multiple CPUs in addition to parallel streams. In each case, the parallel stream range that gives better throughput results is used. In Figure 3.4, there is a significant increase in throughput from single stripe to multiple stripes for LONI and Inter-node results. This indicates that CPU is a source of bottleneck and also excessive usage of stripes and parallel streams cause a drop-down in throughput. For Teragrid, since the results are so close to each other, it is difficult to say that multiple stripes will always give better results based on the network condition. The striping results are obtained using a single node only, yet it is sufficient to reach up to the network throughput limit.

The maximum achievable throughput of inter-node transfers are 9Gbps and this value is gained using only 1 stream and 2 stripes(Figure 3.4). When we look into the CPU Utilization(Figure 3.5) for 2 stripes, we have seen that two cores are at their maximum utilization which is even among these cores. An average of 30% is utilized for 8 cores. But since the NIC throughput limit is reached (only 90% of NICs are utilized in general) the throughput becomes stable around 9 Gbps. Comparing to the 7.5Gbps maximum throughput achieved with single stripe, using multiple CPUs is definitely an improvement.

3.4 Step 4: Effect of Parallel Read& Write on Disk Throughput and Removal of Disk Bottleneck

Disk is usually the slowest part in an end-to-end data transfer. To provide faster data access speed, multiple disks can be used in parallel. In this section, we analyze different access methods to data that

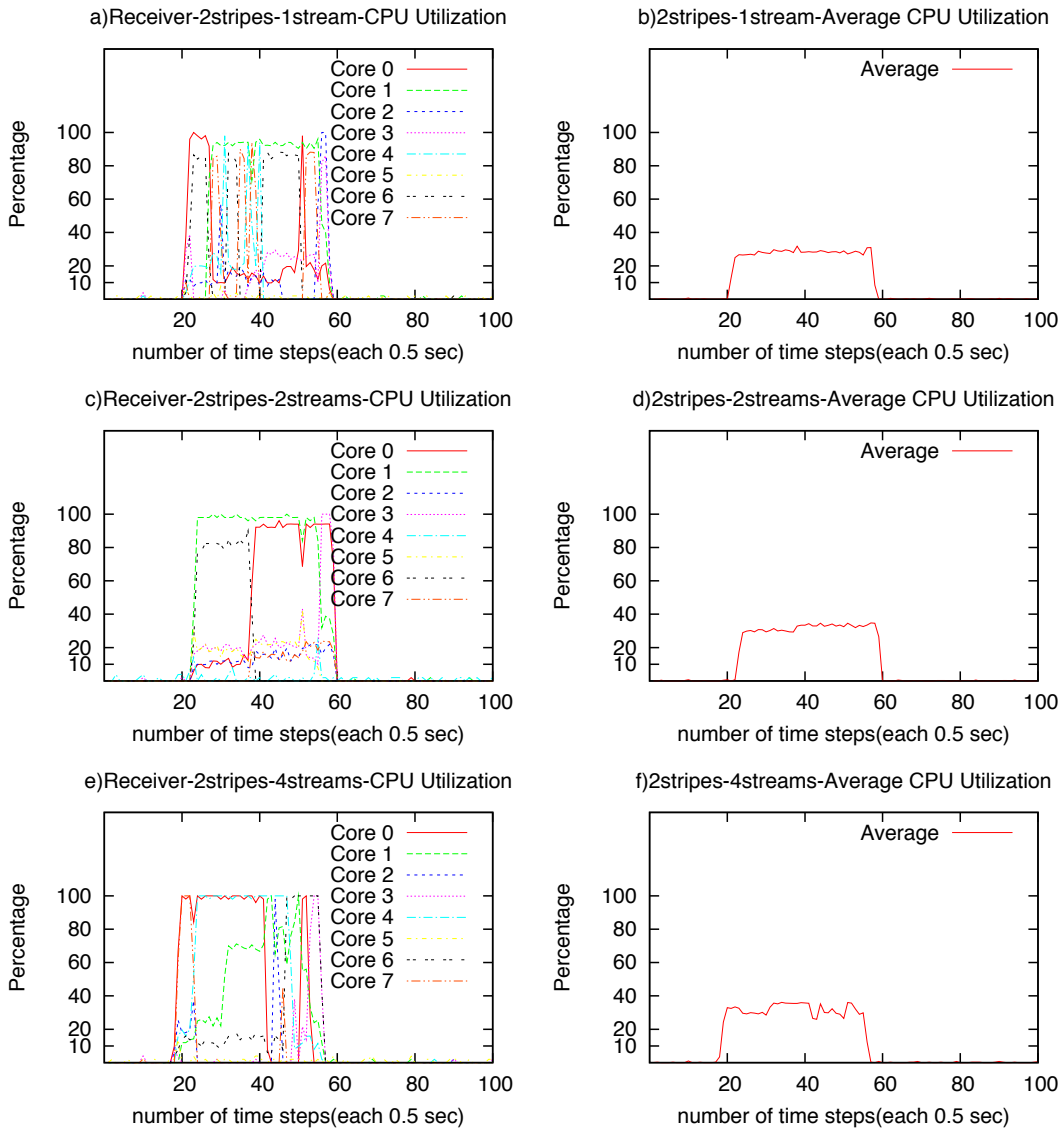


FIGURE 3.5: Effect of Stripes on CPU Utilization on Inter-node GridFTP Throughput

is managed by parallel file systems by using MPI-IO [9] and compare it to a local file system speed of a local hard drive of a node in Spider cluster. Parameters such as data size, stripe number and access method affect the disk throughput. The experiments are conducted over two resources(Abe, Oliver) in Teragrid and LONI managed by Lustre [13], which is a very common parallel file system and another resource(Spider) with local disk access. The access methods used are categorized into two: single file parallel and multi file parallel. In the first method, the reads and writes are done from different parts of the same file while in the second, multiple files are read and written with serial access to each at the same time.

Figure 3.6 presents the disk throughput results of Lustre file system on NCSA using the Abe cluster nodes. In each node there is a total of 4 processor cores. Each MPI instance reads/writes a data size ranging 8MB-256M. As the processor number is increased, the total amount of data read or written is increased as well. For single file parallel write, small data size write speed increases as the number of processors is increased and the throughput reaches up to 6Gbps. However for large data sizes such as 100M and 256M only a slight improvement of throughput is obtained. After 4 processors it starts to decrease. The single file parallel read results are the most unstable results. Large reads start with a high access speed but as the number of processors goes up, the throughput goes down. But for small data sizes(8MB), single file parallel reads make an improvement in terms of throughput going from 2Gbps to 4Gbps. The real benefit of the parallel file system is seen when we write to or read from different files at the same time. In Figure 3.6.c the throughput goes up to around 6Gbps even for large data sizes such as 100M and 256M. For small sizes it even reaches to 16Gbps. The multi file parallel read results do not depend on the data size and they provide similar throughput values and as the number of processors increases the read throughput continues to increase and reaches 60Gbps. Further increase in the parallelism level could increase this value more.

To define generic characteristics for the parallel access disk throughput we conducted the same experiments on a different cluster in LONI network. Figure 3.7 represents the MPI-IO results for Oliver cluster. Similar trends are seen in the throughput curves for Oliver cluster Lustre file system except for the single file parallel read results. However, there are slight differences in the throughput achieved.

The local file system experiments are conducted over the local hard drive of the Spider21, which is an I/O node of the Spider cluster at CCT. The node has a total of 8 processor cores. In Figure 3.8, we see that the single file parallel write speed falls down dramatically for large data sizes when we increase the processor number to 2 and similar results are seen for multi file parallel writes as well. This indicates the difference between a parallel file system and a local file system. The small data sizes write speed is increased for both tests but the increase is much steeper in multi file parallel write. For reads, the data size is not important and the read speed goes up for both tests as the number of processors is increased.

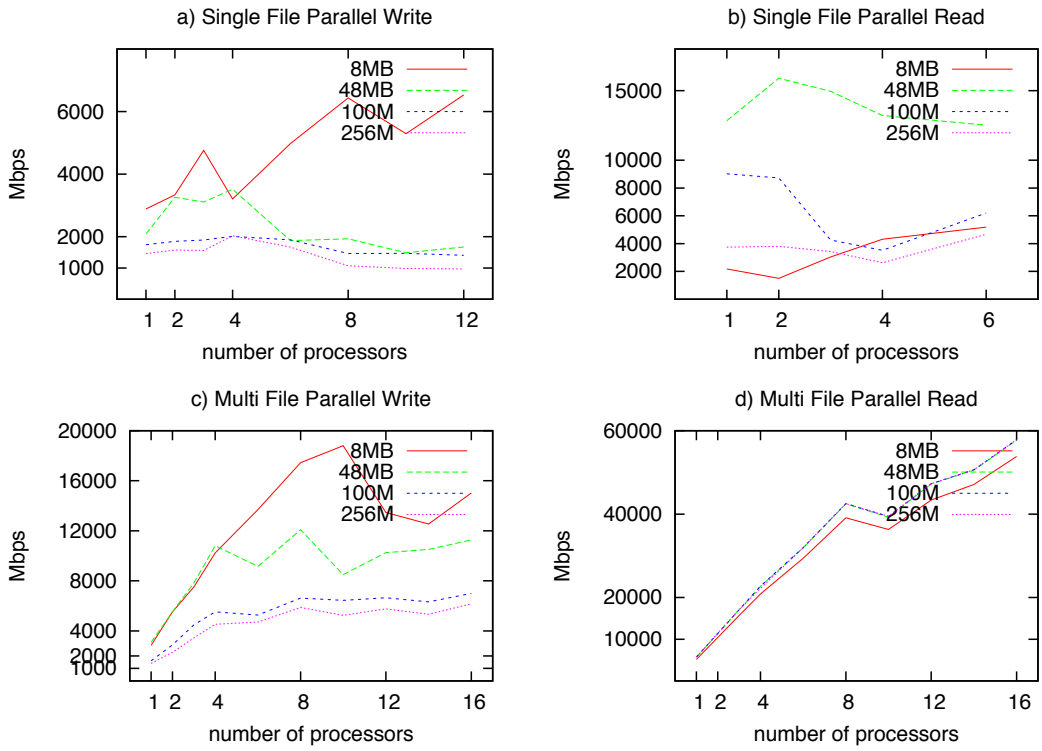


FIGURE 3.6: Abe cluster (NCSA) MPI-IO disk throughput

The results show that the disk throughput is unpredictable and it depends on several factors such as the data size, the processor number, the disk deployment, the access method (serial/parallel), access type (read/write) and the file system.

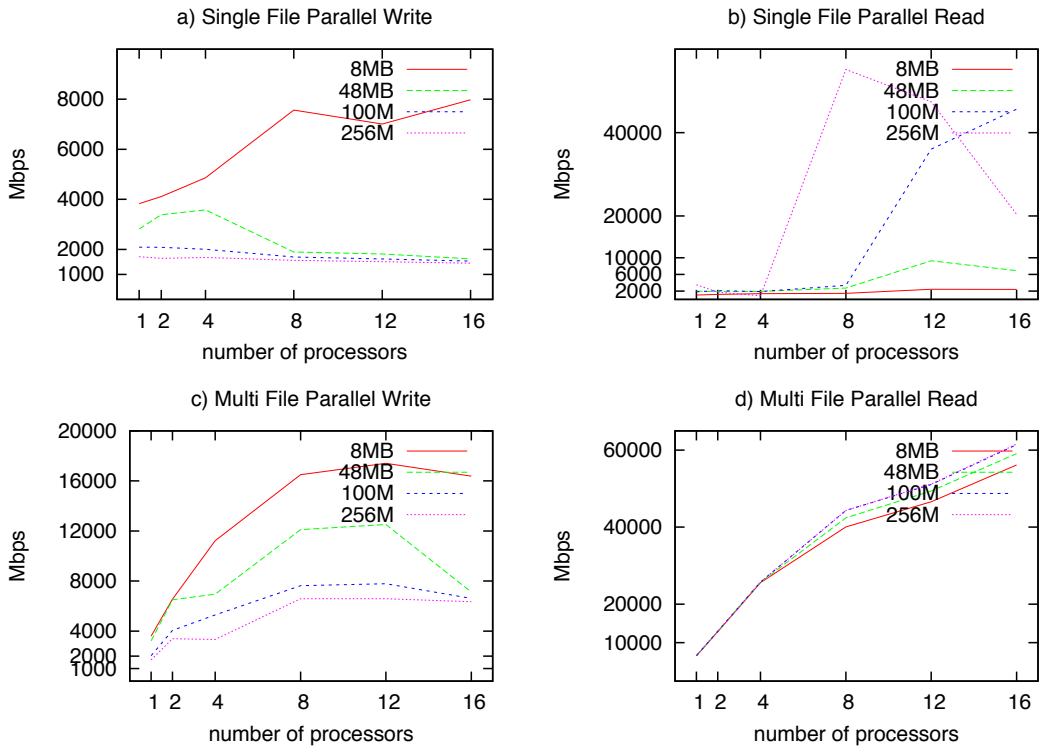


FIGURE 3.7: Oliver cluster (LONI) MPI-IO disk throughput

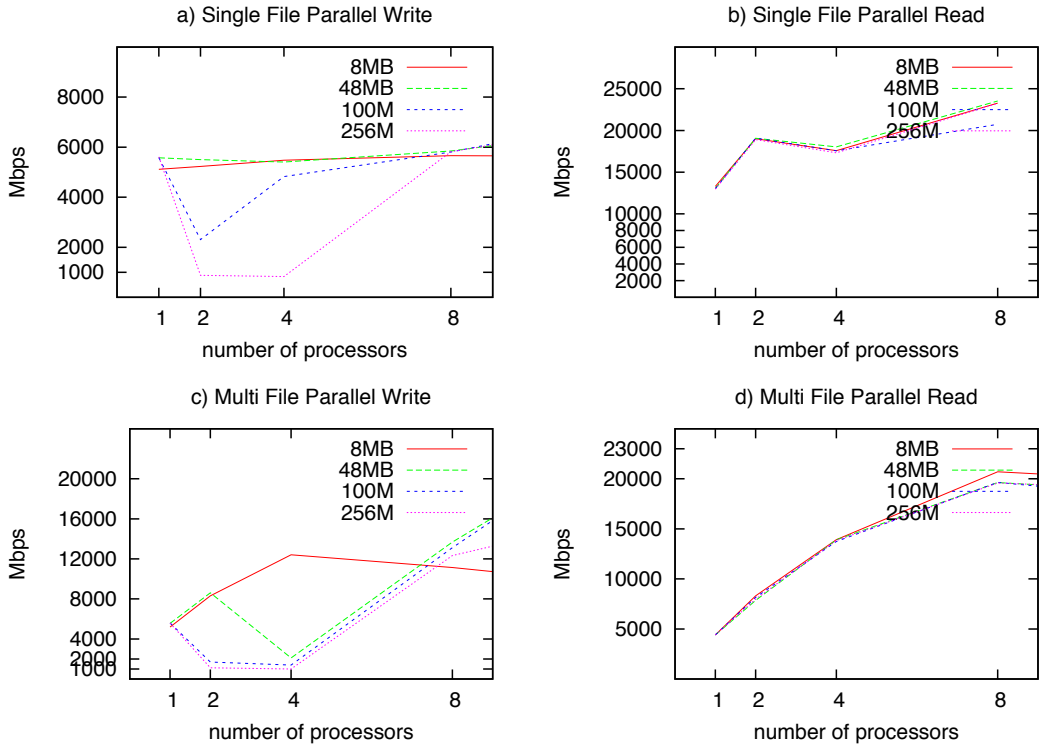


FIGURE 3.8: Spider21 (CCT/LSU) MPI-IO disk throughput

Chapter 4
Models

In this chapter, we present several models for parallel stream optimization and balancing buffer size with parallel streams that we have developed. By using the presented models, we define a flow model that also takes into account the capacities of end-system resources and provide the optimal stripe, node and parallel stream number.

4.1 Parallel Stream Optimization Models

We have developed models that will predict the behavior of parallel streams and tested them in various network scenarios to prove their correctness. The models are adaptive to the network and end-system environment and do the predictions with few historical information or immediate prediction information that is obtained from network performance prediction tools. Even the most easy-to-apply models have a poor prediction accuracy and the others require a lot of information. We have developed several models that are both based on Dinda et al [21] and Hacker et al [10] model that can predict the behavior of parallel streams in a more accurate approximation. In this section, we present these new models and compare and discuss their derivations and implications.

4.1.1 Modeling Packet Loss Rate

The shortage of Hacker et al [10] Model is that there is no information on when the point of congestion will occur as a result of opening multiple streams. The reason of that conclusion is that the behavior of the packet loss rate as the number of streams increase is unpredictable. However, if we can find a model to characterize the packet loss rate, then we can use Equation 2.1 to calculate the throughput gained by n streams.

To achieve this, a methodology similar to the one in Dinda et al Model [21] can be used. However, this time we can not use a partial second order polynomial to model the packet loss rate, since it increases exponentially as the throughput increases logarithmically. By changing the places of Th_n and p_n in Equation 2.1 we get Equation 4.1.

$$p_n = \frac{MSS^2 c^2 n^2}{RTT^2 Th_n^2} \tag{4.1}$$

In this case, we define a new variable Th'_n and correlate it to RTT, MSS, n and Th_n. Hacker et al. proposes that throughput increases linearly in uncongested networks as the number of streams increases however this is not true for congested networks. The throughput achieved by n streams increases logarithmically, so we define Th'_n in the following equation:

$$Th'_n = \frac{RTT_n^2 Th_n^2}{MSS^2 c^2} = a' n^{1/x} + b' \tag{4.2}$$

Ranging x between 2 and a greater number, we investigate how sharp the increase in packet loss will be after the point of congestion. By placing Th'_n in Equation 4.1, we get the following equation for the packet loss of n streams:

$$p_n = \frac{n^2}{Th'_n} \tag{4.3}$$

Considering we can gather the packet loss rates of transfers with two different stream numbers p_{n_1} and p_{n_2} we can find the values of a' and b'.

$$a' = \frac{\frac{n_2^2}{p_{n_2}} - \frac{n_1^2}{p_{n_1}}}{n_2^{1/x} - n_1^{1/x}} \tag{4.4}$$

$$b' = \frac{n_1^2}{p_{n_1}} - a' n_1^{1/x} \tag{4.5}$$

After finding the value of p_n, we can easily calculate the throughput value of n streams by using Equation 2.1 in Hacker et al Model. However this value represents an upper bound on the throughput achieved.

4.1.2 Increasing Curve Fitting with More Information

The study in [21] shows that the characteristic of a throughput curve increases sharply then becomes stable until it reaches the capacity of the link therefore the model represented fits to the data presented. However in real experimental environment by using actual file transfer protocols(e.g. GridFTP) the results could be different from the proposed situation. Figure 4.1 presents a file transfer of 512MB with GridFTP over a wide area network using up to 40 parallel streams. As the number

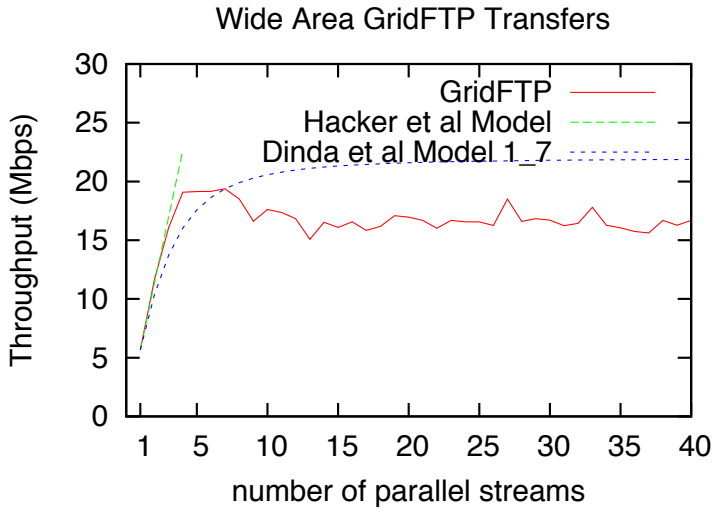


FIGURE 4.1: The aggregate throughput results of GridFTP transfers of a 512MB file over 155ms latency wide area network

of streams increases, the throughput achieved also increases. However, after some point, the created congestion by opening too many streams causes a decrease. The peek point in this case gives us the optimum number of streams. The existing models can not predict this behavior. Hacket et al Model predicts the throughput correctly up to the point where the congestion starts and packet loss rate starts to increase. Dinda et al Model can predict the throughput behavior correctly up to the point the throughput starts to decrease, however it can not predict the decreasing part of the throughput curve.

Instead of using two throughput measurements for two different parallelism levels, we plan to increase this information level into three measurement values. However the overhead of gathering extra information must not surpass the actual speed up gained by opening parallel streams. In this case, we may apply two different methodologies to make use of the extra information. First the calculated a' and b' for using parallelism levels n_{12} and n_{13} are averaged either by using arithmetic, geometric or quadratic averaging methods. Then throughput can be calculated with the averaged values of a' and b'. Second, as we can see from Figure 4.1, the throughput curve acts as two different functions. Until reaching the peek point, it acts as a certain function. However, when falling down it shows a different characteristics. So, instead of using a single function we could break the function into two. By using parallelism level n_1 and n_2, we could model a certain function and by using n_2

and n_3 we could model the second part. Passing between two functions can be a little sharp however this indicates that the optimal number of streams is certainly between n_2 and n_3. In the next section, we present a model that smooths out this sharp transition between two functions.

4.1.3 Logarithmic Modeling of Throughput Curve

The relationship between p, RTT and n is modeled with a partial second order polynomial equation in[21]. However, in some of the cases a linear or full second order polynomial equation may give better results.

We know that the packet loss rate increases exponentially. However, we may not know the order of the equation to use. In this case, instead of using a partial second order polynomial, we use an exponential equation whose order may change depending on the a' and b' parameters. The following equation is used to define the variable p'_n we have mentioned before:

$$p'_n = a'e^{b'n} \tag{4.6}$$

and

$$Th_n = \frac{n}{\sqrt{a'e^{b'n}}} \tag{4.7}$$

By using two throughput measurements Th_1 and Th_2 with different parallelism levels n_1 and n_2 the following values are calculated for a' and b':

$$a' = \frac{n_1^2}{Th_{n_1}^2 \times e^{b'n_1}} \tag{4.8}$$

$$b' = \log_{e^{n_1-n_2}} \frac{Th_{n_2}^2}{Th_{n_1}^2} \times \frac{n_1^2}{n_2^2} \tag{4.9}$$

4.1.4 Dynamic Extraction of Model Equation Order by Newton's Iteration

The discussion about the type of equation models led us to the conclusion that the order of the equation should be extracted dynamically. The logarithmic modeling of throughput that is explained in the previous section is able to make a smooth transition from the increasing to the decreasing part of the prediction curve. However, as the stream number increases the prediction curve approaches to 0 and may not give an approximate prediction result to the actual throughput in the decreasing part

of the curve. To be able to make a good prediction, we have formulized the p'_n by addition of a new variable to predict the order as follows:

$$p'_n = a'n^{c'} + b' \tag{4.10}$$

In this case, c' is the unknown order of the equation additional to a' and b'. Thus our throughput formulation becomes as:

$$Th_n = \frac{n}{\sqrt{a'n^{c'} + b'}} \tag{4.11}$$

To solve this equation, we need three measurements Th_{n_1}, Th_{n_2} and Th_{n_3} on the throughput curve for stream values n_1, n_2 and n_3. Also c' being to the power n makes the solving of the equation much harder. After several substitutions we come up with the following equations for a', b' and c':

$$\frac{n_3^{c'} - n_1^{c'}}{n_2^{c'} - n_1^{c'}} = \frac{\frac{n_3^2}{Th_{n_3}^2} - \frac{n_1^2}{Th_{n_1}^2}}{\frac{n_2^2}{Th_{n_2}^2} - \frac{n_1^2}{Th_{n_1}^2}} \tag{4.12}$$

$$a' = \frac{\frac{n_2^2}{Th_{n_2}^2} - \frac{n_1^2}{Th_{n_1}^2}}{n_2^{c'} - n_1^{c'}} \tag{4.13}$$

$$b' = \frac{n_1^2}{Th_{n_1}^2} - a'n_1^{c'} \tag{4.14}$$

The derivation of a' and b' all depends on c'. To solve the first equation, we applied a mathematical root finding method called Newton's Iteration method. We revised the method to be suitable to our own problem:

$$c'_{x+1} = c'_x - \frac{f(c'_x)}{f'(c'_x)} \tag{4.15}$$

According to that method, after $x + 1$ iterations we are able to find a very close approximation to c'. Starting with a small number for c'_0 we continued to calculate through c'_{x+1}. The value of the most approximate c' depends on only $f(c')$, in this case the first equation above, and its derivative. After calculating a most approximate c' which is possible with only a few iterations, the value of a' and b' can easily be calculated.

4.1.5 Full Second Order Model

The study in [21] briefly compares the partial second order with linear and full second order models. However, the results are compared based on the increasing and then becoming stable characteristics of the throughput and suitable parallelism levels were not used. We believe that a full second order model can predict the increasing and then decreasing characteristics of parallel stream throughput as the number of streams increases.

Regarding to the full second order model, we assume that p'_n is related to a full second order polynomial, other than the partial second order one which was presented before. Adding the linear term to the model will result in a series of changes to the corresponding equations. The following equations are derived to be used in this model.

$$p'_n = p_n \frac{RTT_n^2}{c^2 MSS^2} = a'n^2 + b'n + c' \tag{4.16}$$

According to Equation 4.16, we derive:

$$Th_n = \frac{n}{\sqrt{p'_n}} = \frac{n}{\sqrt{a'n^2 + b'n + c'}} \tag{4.17}$$

In order to obtain the values of a', b' and c' presented in Equation 4.17, we need the throughput values of three different parallelism levels($Th_{n_1}, Th_{n_2}, Th_{n_3}$) which can be obtained from the predictions of network measurement tools or past data transfers .

$$Th_{n_1} = \frac{n_1}{\sqrt{a'n_1^2 + b'n + c'}} \tag{4.18}$$

$$Th_{n_2} = \frac{n_2}{\sqrt{a'n_2^2 + b'n + c'}} \tag{4.19}$$

$$Th_{n_3} = \frac{n_3}{\sqrt{a'n_3^2 + b'n + c'}} \tag{4.20}$$

By solving the following three equations we could place the a',b' and c' variables to Equation 4.17 to calculate the throughput of any parallelism level. Based on equations 4.21,4.22 and 4.23, the values of a',b' and c' can be calculated easily.

$$a' = \frac{\frac{\frac{n_3^2}{Th_{n_3}^2} - \frac{n_1^2}{Th_{n_1}^2}}{n_3 - n_1} - \frac{\frac{n_2^2}{Th_{n_2}^2} - \frac{n_1^2}{Th_{n_1}^2}}{n_2 - n_1}}{n_3 - n_2} \tag{4.21}$$

$$b' = \frac{\frac{n_2^2}{Th_{n_2}^2} - \frac{n_1^2}{Th_{n_1}^2}}{n_2 - n_1} - (n_1 + n_2)a' \tag{4.22}$$

$$c' = \frac{n_1^2}{Th_{n_1}^2} - n_1^2 a' - n_1 b' \tag{4.23}$$

4.1.6 Experimental Results

In this section, we present our test results regarding GridFTP transfers and give prediction results of several methods used. First, we have tested the models in wide area using randomly chosen parallelism levels. Our testbed consists of two Linux machines, one is a Redhat workstation in the Center for Computation and Technology at Lousiana State University, the other belongs to a Suse cluster in University of Trento in Italy. Both machines have installed Globus Toolkit. Also for packet behavior measurements, we used a Linux network analyzer tool called Wireshark.

We have conducted GridFTP transfers with a file size of 512MB. The number of streams ranged between 1 and 40. The results presented in Figure 4.1 are the average of multi-iteration of tests. The actual file transfers by means of a protocol, in our case GridFTP, follows a different characteristics than presented in study [21]. In this case, the aggregate throughput achieved by parallel streams increases as the number of streams increases, after reaching its peek it presents a wavy behavior, however further increasing of stream number results in a descent in throughput. The current models can not predict this unstable behavior.

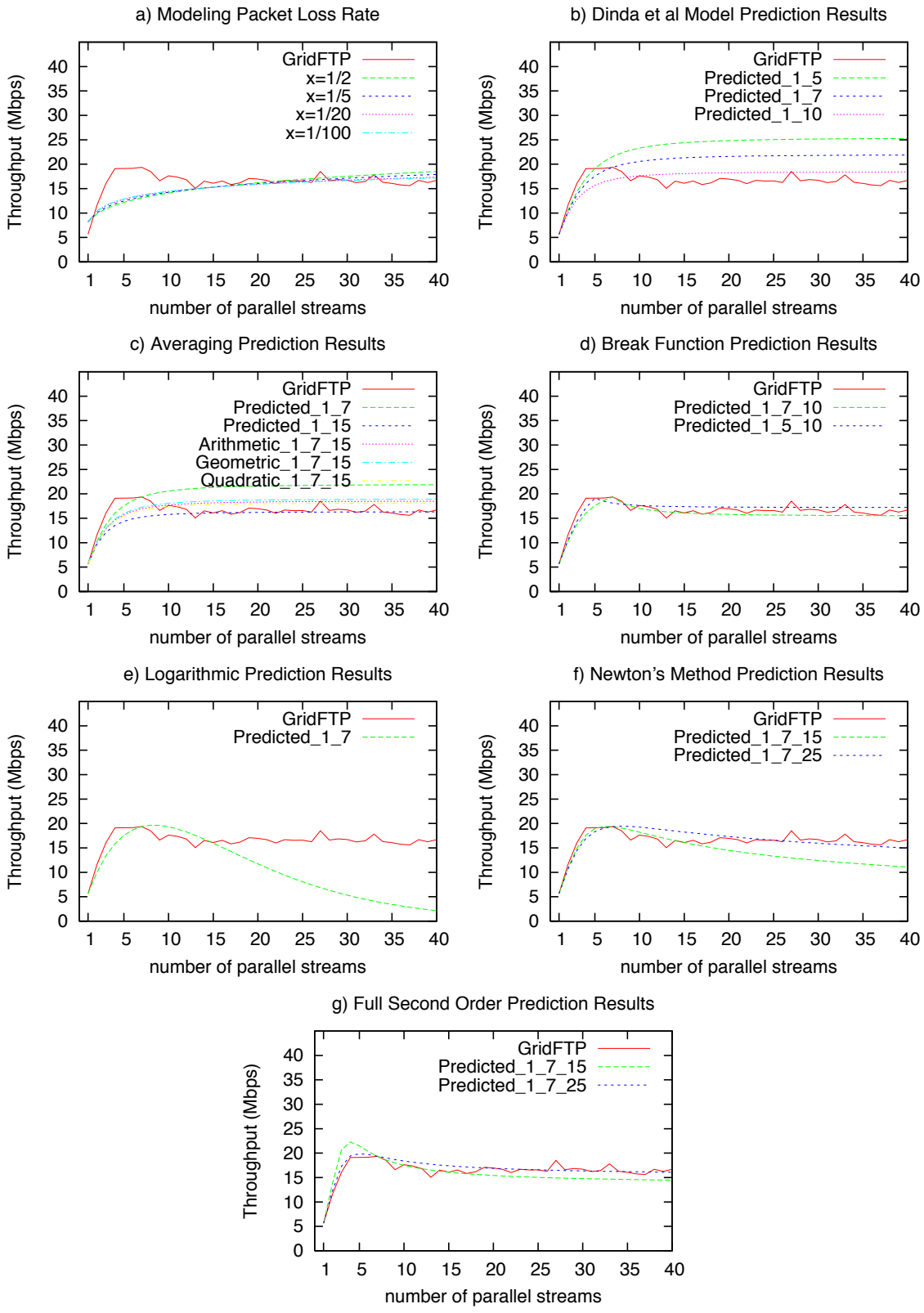


FIGURE 4.2: The prediction results of GridFTP transfers with randomly chosen stream levels

Starting with the shortcomings of Hacker et al Model, the idea of the total aggregate throughput of parallel streams being equal to the throughput of a single stream times the number of streams seems to be proven in uncongested networks. Figure 4.1 shows that up to 3 streams this equivalence is true more or less, however after that point the linear ascent of throughput curve turns into a logarithmic one indicating the existence of congestion in the network. The primary reason of this behavior is the change in packet loss rate due to congestion. By modeling packet loss according to some measured packet loss rates of different parallelism levels taken from Wireshark [33] with our presented model and applying Equation 4.1 we get a predicted packet loss rate and replace it in Equation 2.1 to calculate throughput. As a result, we get the following results presented in Figure 4.2.a. The difficulty of applying this model is that we need a lot of information like MSS, RTT and packet loss rate. Also the resultant value gives us an upper bound on throughput. As we can see from the figure instead of a linear increase which is not suitable with behavior of GridFTP results we can get a logarithmic increase which better suits assuming that packet loss rate increases exponentially as we have proposed in our model. By increasing the x value we could get a sharper knee and a flatter continuous behavior for the rest of the stream numbers.

We also implemented Dinda et al Model and all of the improvements based upon it for randomly chosen parallelism level information. Figure 4.2.b represents the prediction results of Dinda et al Model for different parallelism levels. The prediction curve calculated with throughput results of 1 and 5 parallel streams presents higher results than the ones calculated with 1 and 7, and also 1 and 10. The closest results given for low number of streams are 1 and 5, and also for high number of streams 1 and 10. All of the prediction curves have a logarithmic ascent behavior and then becomes stable for high number of streams but never fall down. The study in [21] mentions that best parallelism values to use are either 1-5 or 1-10. However, we could only tell this after some experiences with the transfers. Instead of that we could try to increase our information level by using 3 parallelism levels. According to our approach, we may detect the behavior change in the function by calculating average a' and b' values. The results of this model is given in Figure 4.2.c. The averaging results give that best stream number values to predict this curve is between 7 and 10. However we do not need to

know exactly this number by using our averaging approach. The averaged curve gives a closer result to both the increasing and decreasing part of the GridFTP throughput curve.

In this case, we know that the throughput curve of GridFTP behaves like two different functions for ascending and descending behaviors. Hence we could use the information regarding three parallelism levels and model the curve by breaking into pieces. Figure 4.2.d shows the prediction results taken by 1-5-10 and 1-7-10 parallelism levels. We could see that the prediction curve almost perfectly suits the GridFTP curve. Although the transitions are a little sharp between the pieces of the curve that gives us an exact idea where the optimal parallel stream number lies. In another model improvement, we have modeled packet loss rate as an exponential function and hence throughput as a logarithmic function (shown in Figure 4.2.e). The transition between the ascending and descending part of the throughput curve is smoothed out. However, the descending part of the throughput curve can not be predicted well as the curve approaches to 0 as the number of streams increases further. We have solved the problems of both the 'break function' and 'logarithmic prediction' methods by predicting model equation order dynamically. Figure 4.2.f shows the results of the predicted throughput by using dynamic equation order extraction by Newton's Iteration mentioned in Section 4.1.4. The best results are taken with 1,7 and 25 parallelism levels although 1,7 and 15 give pretty close results. We are able to predict the actual GridFTP curve with a smooth transition between the increasing and decresing part of the curve and with a good approximation overall. With this method the peek point of the curve will give us the optimal stream number to open for maximum throughput. In the last model, we have applied a full second order equation and have seen that if the correct parallelism levels are used this model predicts the throughput value precisely. Hence, although 1-7-15 gives exaggerated results for the maximum point in throughput, 1-7 25 gives accurate results. In this case, it becomes an important decision which paralelism levels to use to get the most accurate results.

The best results were taken with Dynamic Equation Order(Newton's Iteration Method) Model and Full Second Order Model. To prove the correctness of our models, we increased the variety of our testbed. We have tested the two models both in high-speed networks and low-speed(100Mbps) LAN and WAN. Our new testbed consists of four 256-processor-clusters in the LONI network [36], two workstations in DSL Laboratory and CCT and a workstation in University of Trento in Italy. We

have divided our test cases into 5 categories: LONI-LONI, LONI-LAN, LAN-LONI, LAN-LAN and LAN-WAN. For the first case we have used two clusters from the LONI network with 1 Gbps and 10Gbps NICs, for the second case we have selected a cluster in the LONI network and a workstation in DSL, third case is a switched version of the second case, for the forth case two workstations in CCT and DSL Lab are used and finally for the fifth case we used two workstations from CCT and University of Trento in Italy. For all the test scenarios we have selected GridFTP protocol to do the transfers.

Each model presented in the previous sections can show their best performances if the sample throughput data to calculate the predicted throughput can be chosen from appropriate parallelism levels. All of the models either need 2 or 3 throughput data of different parallelism levels. Hence there exists many kinds of combinations if we have more than three pairs(n, Th_n) of data and it is important for us to find the best combination to minimize the distance between historical or prediction data and calculated throughput of n streams based on the models presented. However the number of combinations is too large, hence we provide an intelligent selection strategy which decides on less number of data and can be used with an online model as well. Our previous experiences showed that it is better if we choose the parallelism levels not close to each other. So we applied an exponential increase strategy by selecting the stream numbers that are power of 2: $1, 2, 2^2, 2^3, ..., 2^k$. Each time we double the number of streams until the throughput starts to drop down or increase very slowly compared to the previous level. After k+1 steps we gather k+1 parallelism level throughput data. The outline of the sampling strategy is presented in Algorithm 1.

Algorithm 1 Sampling Algorithm

Output: Th_N: a set of throughput values for different parallelism levels from the sampling algorithm
$i \leftarrow 1 \;\; p \leftarrow 1$
$Th_{N_i} \leftarrow getThroughput(p)$
$i \leftarrow i + 1 \;\; p \leftarrow p \times 2$
$Th_{N_i} \leftarrow getThroughput(p)$
while $Th_{N_i} > TH_{N_{i-1}}$ *and* $Th_{N_i} - TH_{N_{i-1}} > precision$ **do**
    $i \leftarrow i + 1 \;\; p \leftarrow p \times 2$
    $Th_{N_i} \leftarrow getThroughput(p)$
end
return i, Th_N

The sampling algorithm provides us with a set of throughput values for exponentially increasing parallel streams. However we only need three data points to apply our prediction model. Another algorithm is needed to choose those data points among the available values(Th_N) and it is presented in Algorithm 2. For all the possible combination of parallelism levels in increasing order, the model is applied and the parameters a',b' and c' are calculated for prediction. The distance between the predicted and actual throughput is calculated which gives us the error value. The combination of parallelism values with the minimum error value is returned.

Algorithm 2 Selection Algorithm

Input: Th_N: a set of throughput values for different parallelism levels from the sampling algorithm
n: number of values in Th_N

```
i ← 1 , j ← i + 1 , k ← j + 1
for i <= n − 2 do
    for j <= n − 1 do
        for k <= n do
            Calculate a′,b′ and c′
            Calculate Predicted_ijk for parallelism levels 1 to N based on a′,b′ and c′
            m = 1
            err = 0
            for m <= N do
                err+= abs(Predicted_m-Th_m)
                m = m + 1
            end
            if err < minerr then
                min_i ← i , min_j ← j , min_k ← k , minerr = err
            end
        end
    end
end
return min_i, min_j, min_k
```

For the first test case, both Dynamic Equation Order(Newton's Iteration Method) Model and Full second order model perform well and slightly better than the Dinda et al model which gives a more smooth transition against the GridFTP curve(Figure 4.3). The curve does not decrease as the number of streams increases because the underlying network is 10 Gbps while the NICs on the systems are 1Gbps. Hence Dinda et al Model can still give good results. On the other hand, the results are quite different when we used 10Gbps NIC cards. Figure 4.4 shows the results of the application of models when the transfers occur between two LONI IBM clusters with 10Gbps NICs.

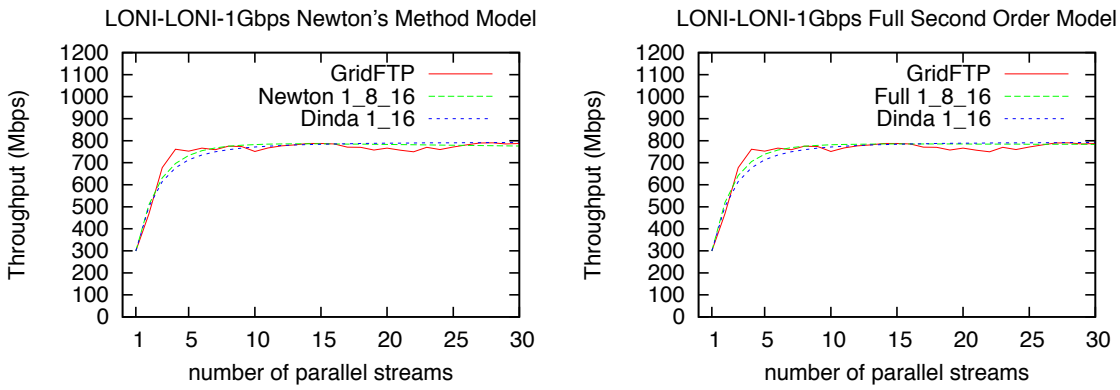


FIGURE 4.3: Comparison of Prediction Models over 1Gbps link

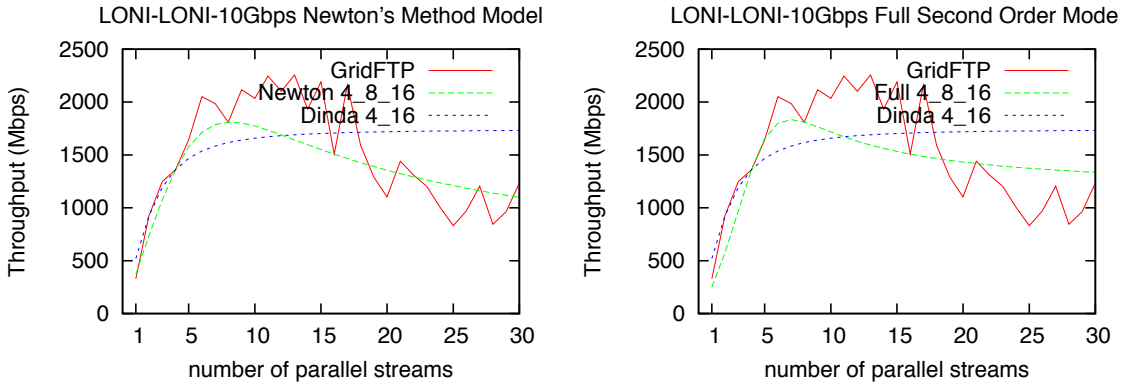


FIGURE 4.4: Comparison of Prediction Models over 10Gbps link

The averaged throughput starts to increase as we increase the number of streams. After going in the same level for a range of stream numbers, it starts to drop down as a result of further increasing the number of streams. Dinda et al model can not predict this behavior. However both Dynamic Equation Order(Newton's Iteration Method) model and Full Second Order model can predict the optimal stream number where the throughput first reaches the optimal point.

In the second test case, all three models performs similar. Here the maximum bandwidth that can be obtained reaches upto 40Mbps limited by the lower system's capabilities(Figure 4.5). In the third case, interestingly the throughput can reach upto 25 Mbps. This again shows us the effect of the end-systems over the network performance. Here Dynamic Equation Order(Newton's Iteration Method) model performs best and Full Second order follows it(Figure 4.6). But Dinda et al Model gives results far from the actual GridFTP curve. For the fourth case, we conducted the experiments in LAN environment and the characteristics of the curve was interesting(Figure 4.7). After falling down

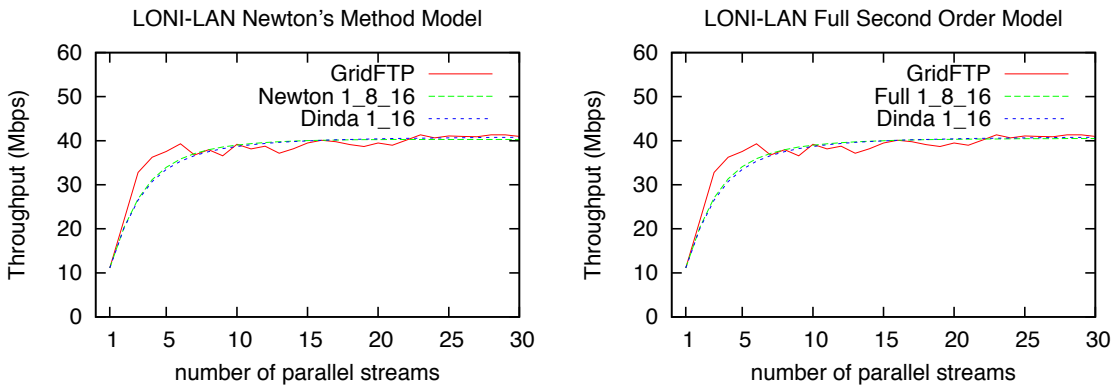


FIGURE 4.5: Comparison of Prediction Models over 1Gbps-100 Mbps link

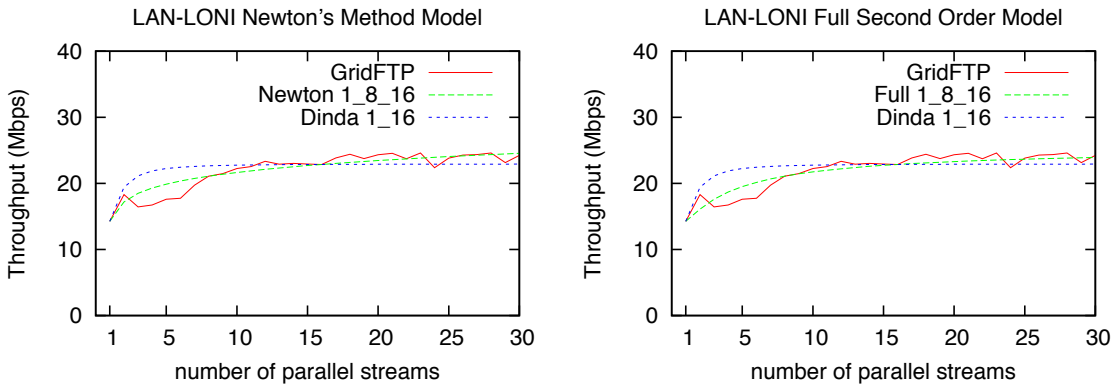


FIGURE 4.6: Comparison of Prediction Models over 100Mbps-1Gbps link

in two streams the throughput started to increase reaching a peak point and then started to decrease coming to a stable point. The Dinda et al Model is far from predicting this unusual behavior. However Dynamic Equation Order(Newton's Iteration Method) Model could predict this behavior with the best accuracy while full second order model misplaces the peak point. In the final case, the same wide area results that were used in Figure 4.2 are used however we apply the intelligent selection strategy of parallelism levels on the prediction results. Again the throughput increased steeply upto reaching the peak point then started to decrease due to congestion in the network(Figure 4.8). Full second order model performed slightly better than Dynamic Equation Order(Newton's Iteration Method) Model and Dinda et al Model was not able to predict the characteristics of the curve.

With these results, we have seen that our models can adapt to different network environments and end-system characteristics and predict the parallel stream throughput behavior accurately.

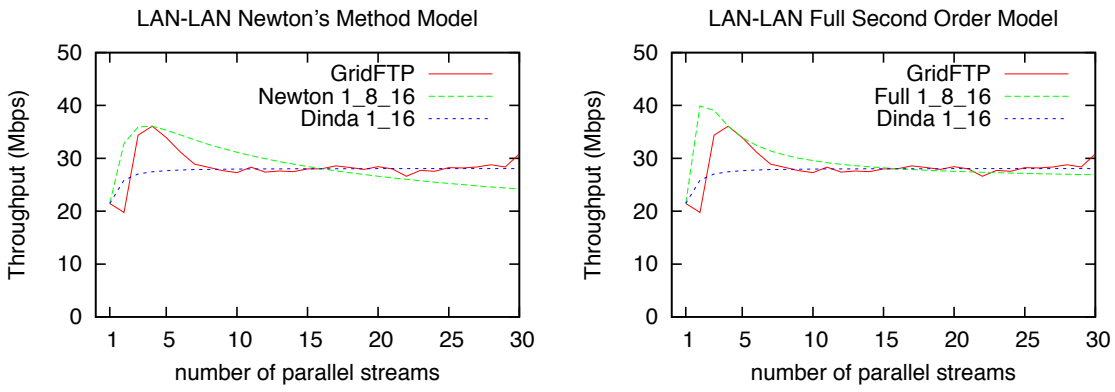


FIGURE 4.7: Comparison of Prediction Models over 100 Mbps local area link

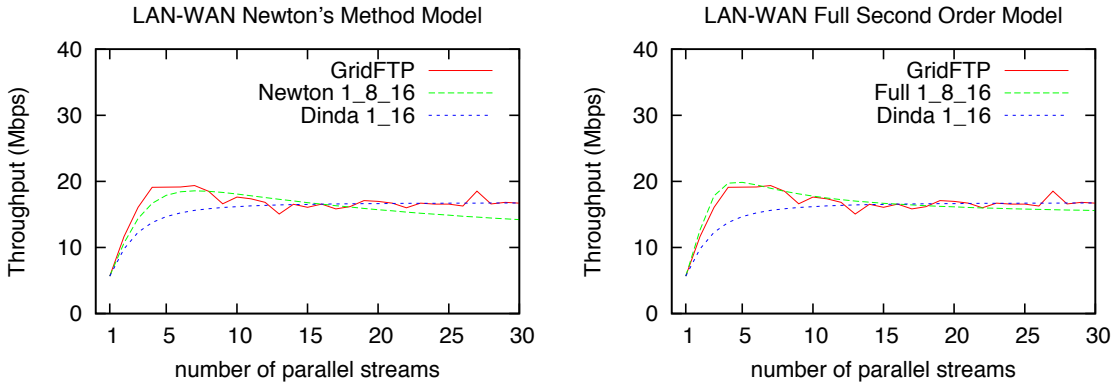


FIGURE 4.8: Comparison of Prediction Models over 100 Mbps wide area link

4.2 Balancing Buffersize vs Parallel Streams

There has been a large number of studies in the area of buffer optimization and our results in parallel stream optimization are very promising. However, a good combination of tuned buffer size and parallel streams could even give more effective results than the single applications of these two techniques. Unfortunately, there is no practical work to balance the buffer size and parallel stream number to achieve the optimal throughput. In this section, we present the results and discussion of simulations performed on NS-2 and a set of experiments in real network environment combined with the application of our models to describe the way a balance must be set. We conducted those experiments in LONI network with two IBM AIX cluster nodes which have 10G NICs and a 6 ms delay between them. The network traffic of the LONI network varies a lot hence we could see the effect of the application of both techniques better.

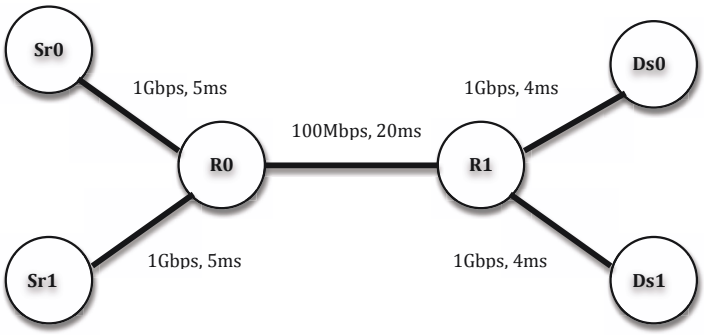


FIGURE 4.9: Topology

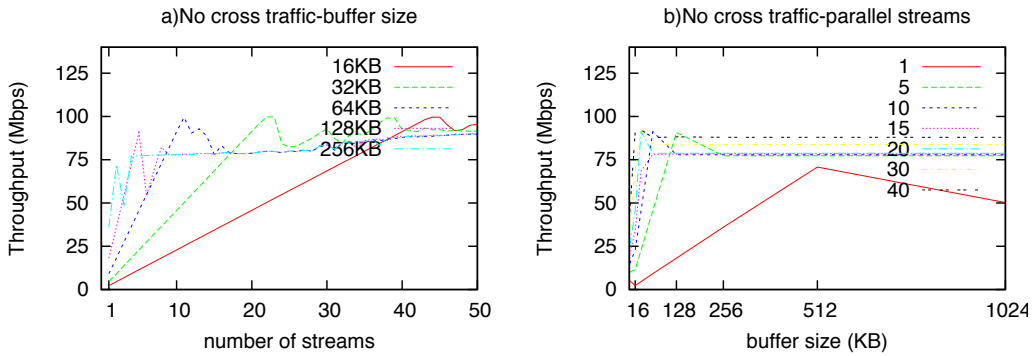


FIGURE 4.10: Effect of parallel streams and buffer size under no cross traffic

4.2.1 Simulation Results and Discussion

In our simulations we have tried different scenarios by changing the buffer size and parallel streams. The network topology we used is represented in Figure 4.9. The bottleneck link bandwidth is 100Mbps with a 20ms delay. The sources($Sr0$,$Sr1$) for our transfers and the cross traffic are connected to the bottleneck link router $R0$ with 1Gbps bandwidth and 5ms delay, while the destination nodes ($Ds0$,$Ds1$) are connected to $R1$ with 1Gbps bandwidth and 4ms delay. The cross traffic flows from $Sr0$ to $Ds0$ while the actual transfer occurs between $Sr1$ and $Ds1$.

In the first set of experiments we did not use any cross traffic and changed the buffer size with parallel streams. With a very small buffer size such as 16KB, the full utilization of the network can be achieved only with a very large number of streams (Figure 4.10.a). After reaching its peak point, the throughput starts to fall down around 45 streams. Increasing the buffer size into 32KB, peak throughput point is pulled back to 22 streams. Further increasing the buffer size to 64KB pulls

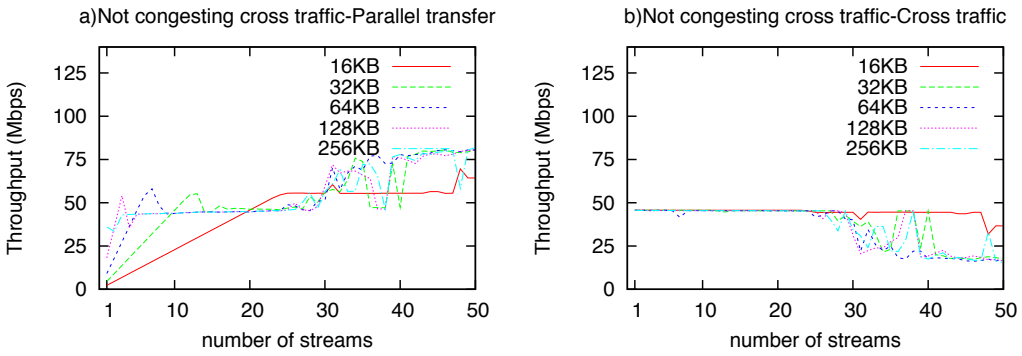


FIGURE 4.11: Effect of parallel streams and buffer size with a cross traffic of 5 streams with 64KB buffer size

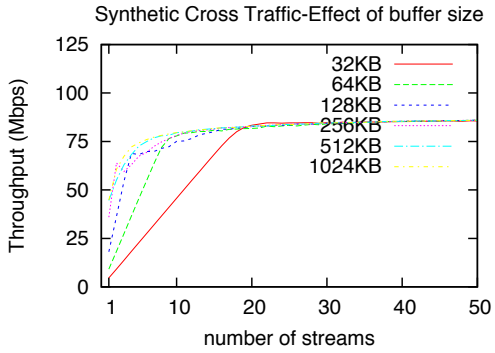


FIGURE 4.12: Effect of parallel streams and buffer size with a randomly generated internet cross traffic

the stream number further to 10 streams. However for larger buffer sizes than 64KB, the throughput never can reach its maximum point. The reasonable selection in this case is to use a buffer size around 16KB-64KB and parallel streams around 45-10 respectively to be able to get the highest throughput. In this case, further increasing the buffer size does not help but causes a decrease in the throughput achieved. The maximum throughput values can be gained with a smaller buffer size than BDP and usage of parallel streams. The figure shows us a wave behavior of throughput, when we increase the buffer size and decrease parallel streams, it eventually decreases in its peak point. In Figure 4.10.b, we compare the parallel streams in vice versa. We could see that larger stream numbers could gain more throughput in smaller buffer sizes.

In the second set of experiments, we have done the simulations in the existence of a non-congesting cross traffic of 5 streams of 64KB buffer sizes. The results were quite interesting. With a very small

buffer size of 16KB, the throughput increases linearly up to the congestion point with the cross traffic as we increase the parallel streams(Figure 4.11.a). The throughput is shared between the two traffics. Further increasing the buffer size will pull the peak throughput point into smaller stream numbers. This kind of behavior is similar to the previous case where there is no cross traffic except that the throughput is shared. In the mean time, there is no effect on the cross traffic as the parallel stream number increases(Figure 4.11.b). However, further increase of parallel stream number in our traffic results in the cross traffic to loose the fight as the total throughput of our traffic starts to increase, the throughput of the cross traffic starts to decrease. The best results of throughput without affecting the cross traffic in this case is 32KB-64KB buffer size with a parallel stream range of 6-13 streams.

In the third case, we used a NS2 library called *PackMimeHTTP* to randomly generate Internet traffic on the bottleneck link. The library generated HTTP requests of random sizes and in the amount of 200 connections per milliseconds. The results are presented in Figure 4.12. According to these results with bigger buffer sizes the maximum throughput point is again reached quicker and with less number of streams.

4.2.2 LONI Experiments

The general idea perceived from the simulation results is that with the usage of tuned buffer sizes and parallel streams we could get an improvement over the maximum throughput that could be achieved. This conclusion was better observed when there was no congesting traffic at all. But with the addition of random traffic, the optimal buffer size and parallel stream values were hard to be captured since the combinations gave us similar maximum throughput values at different points on the curve(Figure 4.12). However in the experiments conducted over the LONI network this difference is much more apparent than the simulation results.

Figure 4.13 shows us the the average throughput results of 5 different buffer sizes combined with a range of parallel stream values between 1-50. When a buffer size of 128KB is used the maximum throughput value is in the range of 15-25 streams and around 2 Gbps(Figure 4.13.a). If we increase the buffer size to 256KB, the maximum throughput is achieved around 10 streams however it could reach upto 2.5 Gbps(Figure 4.13.b). Further increase of buffer size pull back the optimal stream number to smaller values(Figures 4.13.c and .d). However the maximum throughput point falls down

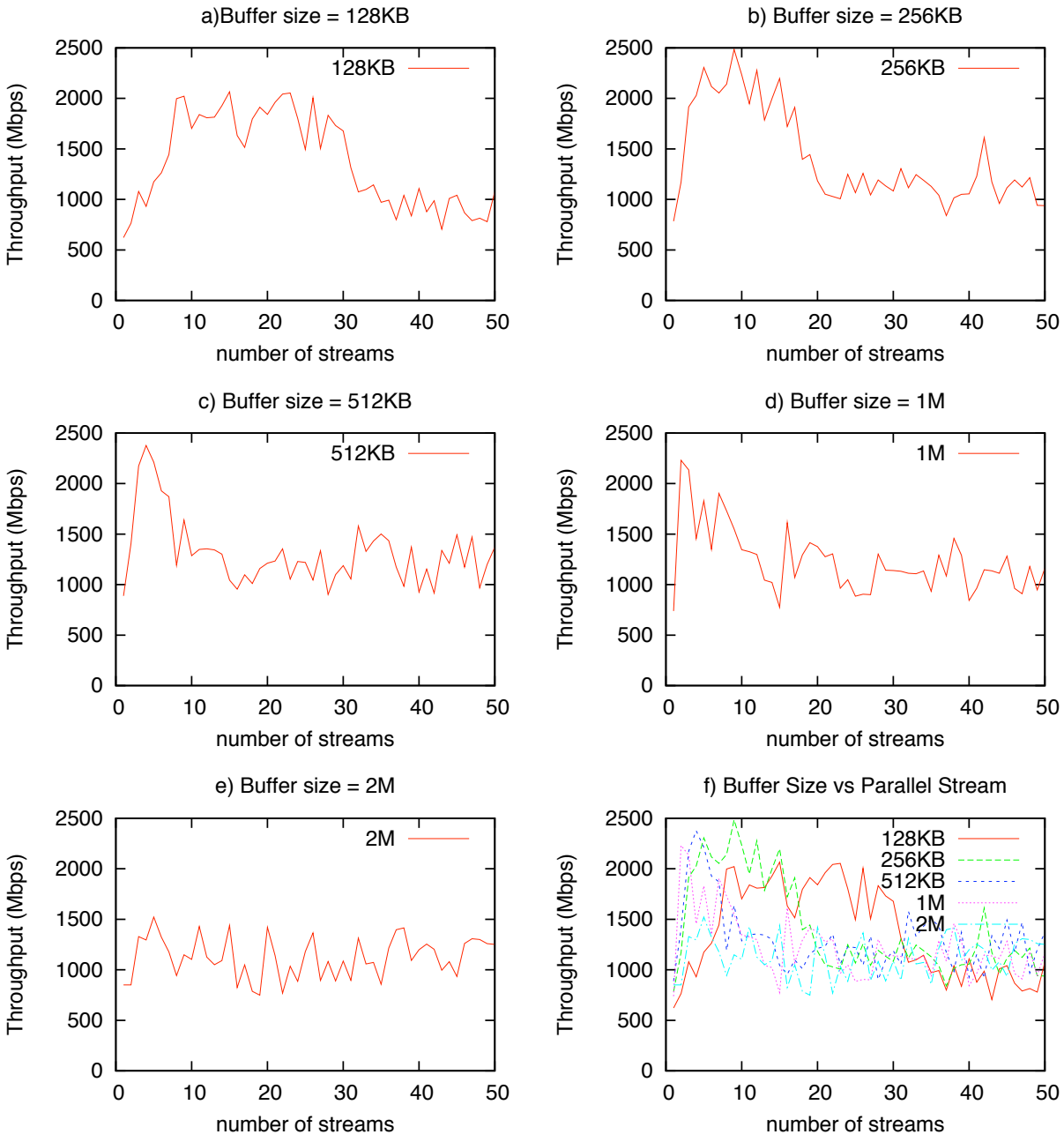


FIGURE 4.13: Effect of parallel streams and buffer size over 10 Gbps LONI network

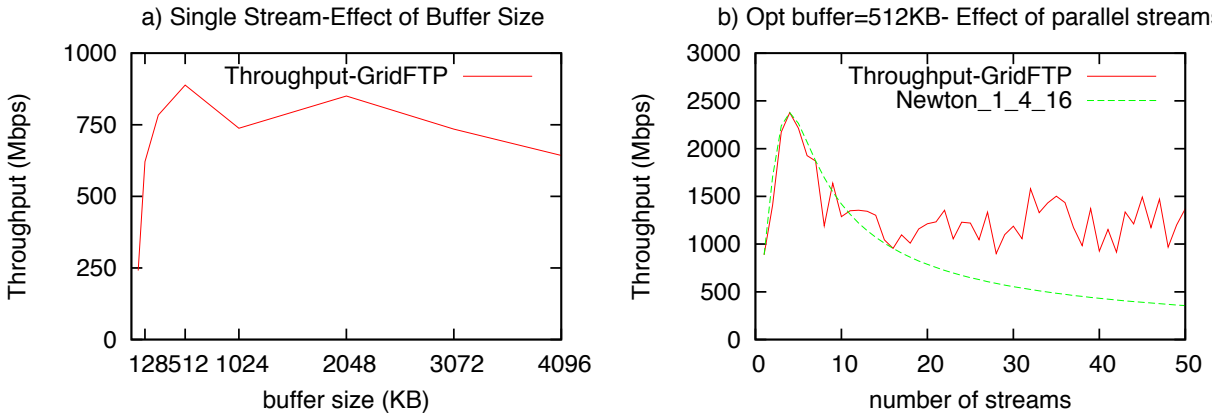


FIGURE 4.14: First technique for balancing buffer size and stream number

to 2.25 Gbps. Further increase results in dramatic falls in maximum throughput and unsteady results(Figure 4.13.e). The whole picture could be seen in Figure 4.13.f . The behavior of throughput is like a wave that rises to a maximum and then falling down to a smaller level as the buffer size ranges. Among the five combinations of curve there is an optimal value of buffer size between 256KB-512KB and a parallel stream value between 5-10. By using these values we could achieve a throughput of 2.5 Gbps in average.

We have considered a rather practical set of approaches to predict the optimal combinations. In the first approach, we conducted some sampling transfers by increasing the buffer size exponentially using only single stream. In Figure 4.14.a, the optimal buffer size that gives the highest throughput is 512KB. By setting the buffer size to this value we have ranged the number of streams between 1-30 and applied the Dynamic Equation Order(Newton's Iteration Method) model(Figure 4.14.b). The optimal stream number that is decided by the model, in this case 4 streams, with a buffer size of 512KB gave us a throughput of almost 2.5 Gbps which is the maximum throughput that was achieved in our previous tests.

As a second approach we reversed the order of the procedure. But for applying the optimization model on parallel streams we have to choose a random buffer size to do the transfers. If we by chance choose the optimal number for single stream and apply the model(Figure 4.15.a) and get an optimal parallelism level, then ranging the buffer size over that value to see if it has any effect, it is shown that the throughput is not improved at all(Figure 4.15.b). On the other hand, if we randomly choose

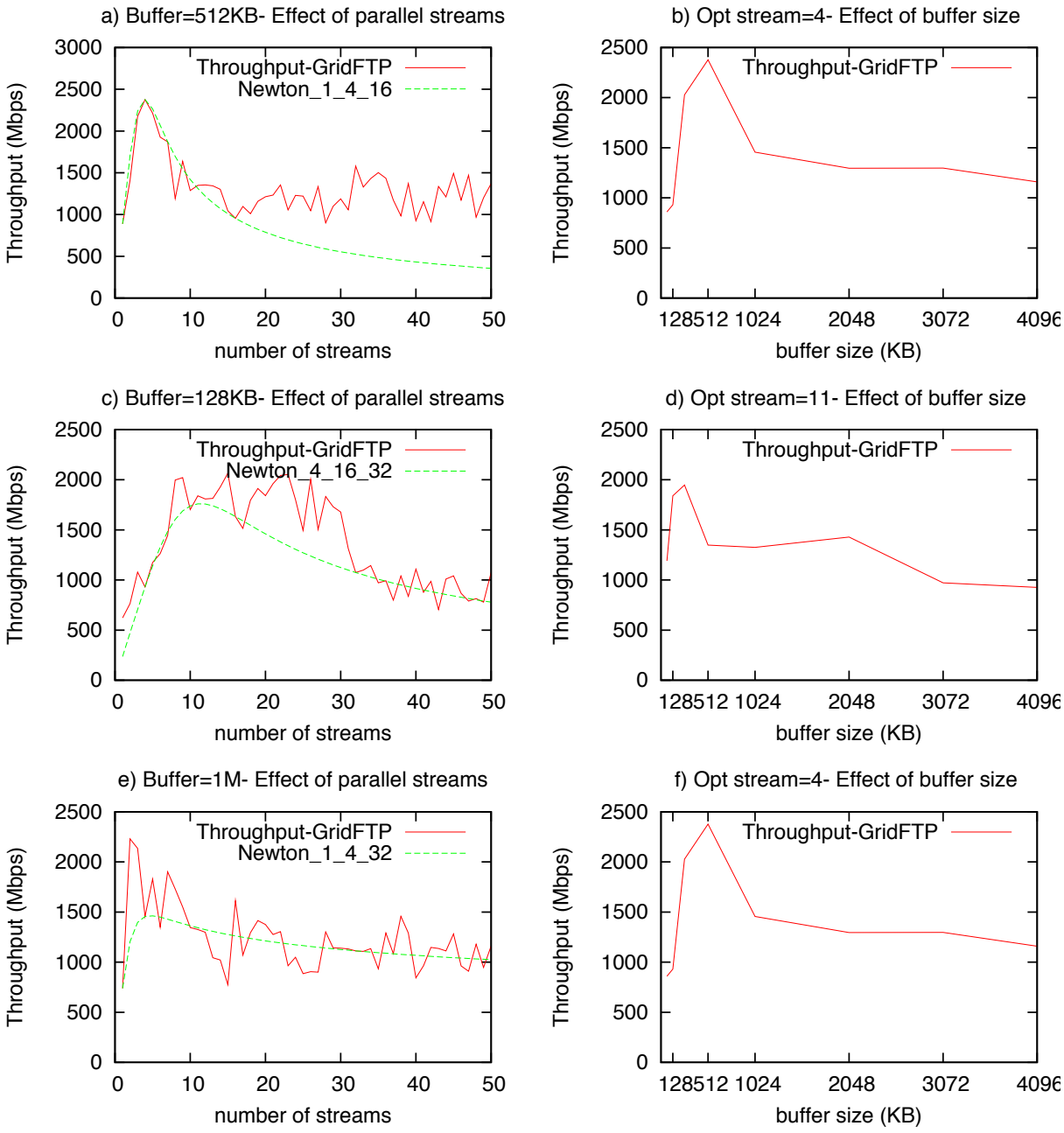


FIGURE 4.15: Second technique for balancing buffer size and stream number

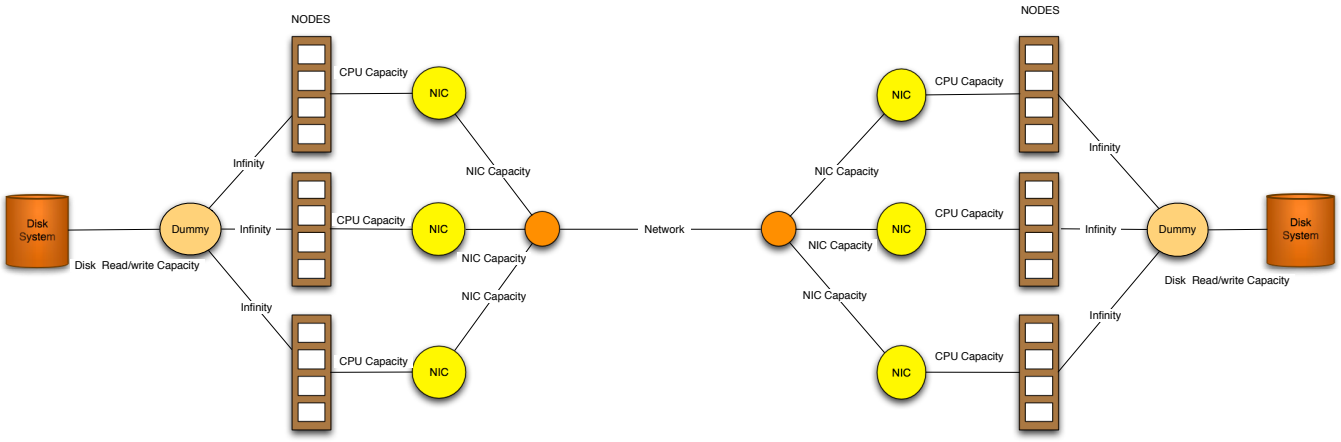


FIGURE 4.16: Flow Model Graph

a non-optimal buffer size and apply the model (Figures 4.15.c and e), we may get an improvement by ranging the buffer size over that throughput as in Figures 4.15.d and f, however we may not always get as high throughput as we can get with the optimal buffer size value. Hence the first technique is a better approach to achieve the maximized throughput.

4.3 Flow Model of End-to-end Throughput

The end-to-end data transfer throughput could be modeled as a specialized version of maximum flow problem. In a maximum flow problem the goal is to send as much flow as possible between two special nodes, without exceeding the capacity of any arc [1]. In Figure 4.16, we present the path of a data transfer between two multi-node clusters. In this model, we consider disk and multi-core nodes of the cluster as a node itself with capacities. The memory-to-memory transfer is modeled with two dummy nodes as source and destination with infinite capacities connected to the cluster nodes. The disk system is linked with one arc because we may not know the actual parallelism level of the file system. While all of the capacities could be measured in terms of throughput unit, the CPU has only usage percentage, in other words utilization. However there is a positive correlation between the CPU utilization and the throughput gained, hence we provide a model that could convert between throughput and utilization percentage.

If we look at the graph in Figure 4.16, we could see that it is a bipartite graph, hence it makes the application of a maximum flow algorithm very easy and there is no need to keep a residual network. It is enough to extract the value of a flow each time the flow is increased. Another issue is that the

outcome of the model will be the number of parallel streams per stripe, the number of stripes per node and the number of nodes. So the parallel stream number should be the same for all the stripes to enable the implementation of the model easier. The following variables are used to define the model:

U_{ij} = Total capacity of each arc from node i to node j (e.g. %100 is the average utilization of all the CPUs of a node and could be related to throughput through regression model)

U_f= Maximal (optimal) capacity of each flow (stripe) (e.g. each GridFTP flow has an optimal throughput of 7.5Gbps with %30 CPU utilization for internode setting in 3.2)

N_{opt} = Number of streams for U_f

X_{ij} = Total amount of flow passing $i->j$

X_{fk} = Amount of each flow(stripe)

N_{S_i} = Number of streams to be used for $X_{fk_{ij}}$

$S_{x_{ij}}$ = Number of stripes passing $i->j$

N_n = Number of nodes

The following inequalities must hold for any algorithm that will be devised:

$$0 \leq X_{ij} \leq U_{ij} \tag{4.24}$$

$$0 \leq X_{fk} \leq U_f \tag{4.25}$$

The biggest challenge to apply this model lies in finding the capacities of each arc. The easiest one is the capacity of the NIC, which could be found by some system command. However our tests show that only %90 of the capacity of NIC is utilized. We have seen that no matter how large the stream number is, in situations when NIC and the network is not a bottleneck the total utilization of CPU never passes %140 and only 2 processors are used out of n processors in a node. In that case in addition to using the capacity of all of the CPUs we also define a flow capacity that should not be exceeded (U_f) . This capacity can be found using the Newton model presented in the previous section. However for disk, it is much more complicated and unfortunately it does not show a deterministic characteristic depending on the number of stripes. As we have mentioned before, it depends on many

factors. The hardest part will be to find the capacity of the disk and network which can only be found via doing sampling by increasing the stream and stripe numbers in a specific methodology considering the capacities of the nodes. In the following sections we present models to do conversions between CPU utilization and throughput and also algorithms to find the optimal parameters for homogeneous and heterogeneous source and destination pairs.

4.3.1 Modeling CPU Utilization with Regression

The experiments with the parallel streams show that there is a high positive correlation between the throughput of the parallel streams and the CPU utilization. To present the maximal achievable end-to-end throughput problem as a flow problem we need to define the CPU Utilization Capacity in terms of the throughput. Henceforth by using the correlation between these entities we provide a regression model that will give us the predicted CPU Utilization curve from the predicted throughput curve. Since there is a linear relation between them we define it as follows:

$$U_{cpu} = a + b \times Th \tag{4.26}$$

U_{cpu} represents the CPU Utilization in terms of percentage while Th represents the throughput. a and b variables are to be found by using the regression model on the available number of data values for CPU utilization and corresponding throughput value of parallel streams. The data numbers used for prediction of the throughput value in Section 4.1.4 is enough to derive the regression model. The variables are calculated according to the following formulas that are derived by using the method of least squares.

$$a = Mean(U) - b \times Mean(Th) \tag{4.27}$$

$$b = \frac{\sum ThU - (\sum U \sum Th/size)}{\sum Th^2 - (\sum Th)^2/size} \tag{4.28}$$

In Figure 4.17.a, we see the application of the Newton's Iteration Model over the throughput of the parallel streams between Spider and Abe clusters using single stripe. While the dots represent the actual data values available, the continuous curve represents the predicted values. By giving

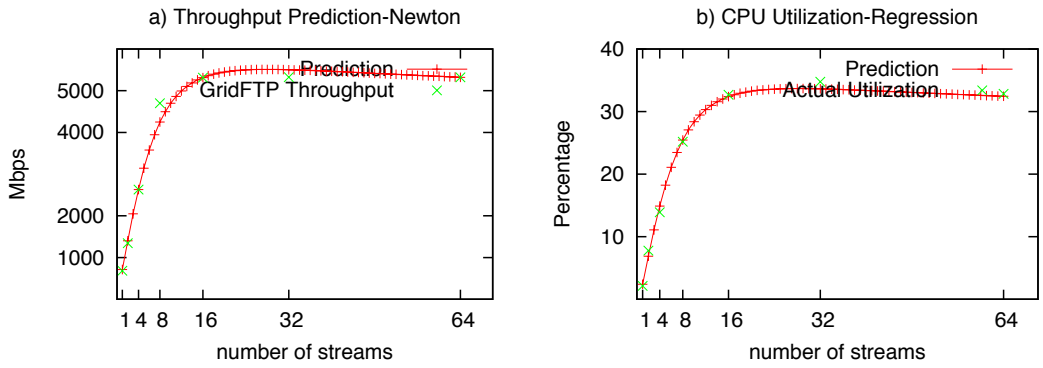


FIGURE 4.17: Regression model of CPU Utilization

those predicted values to our regression model derived from the dotted values, we generate the CPU utilization curve of the parallel streams in Figure 4.17.b. When we compare it to the dots in that figure we can see that we provide a very accurate model for the CPU utilization of parallel streams. In that case the conversion between utilization and throughput is easily done which will give us the advantage to define CPU utilization capacity in terms of throughput units. However we only use the increasing part of the curve to do conversions because sometimes the throughput may fall down although the cpu utilization stays the same.

Chapter 5
Optimization Algorithms

The discussion from Chapter 3 indicates that using only parallel streams is not enough to utilize the large network bandwidth. In certain situations, the CPU, the disk or the network interface could be the source of bottleneck. In this chapter, we present algorithms that will give the optimal number of streams, number of stripes per node and the number of nodes for memory-to-memory and disk-to-disk transfers for an available maximum number of resources for different cases. The algorithms make use of the flow model, the regression model for CPU utilization and the dynamic equation order model presented in Chapter 3 for the optimal number of parallel streams.

5.1 Optimization Algorithm for Unknown Disk and Network Capacity in Homogeneous Resources(OPT_B)

The algorithm takes as input a set of throughput values for exponentially increasing parallelism levels extracted with the sampling algorithm presented in Algorithm 1; the network interface capacity; the destination CPU capacity; and the available number of resources. A number of assumptions are made about the application of the algorithm. The conclusion from Chapter 3 indicates that the destination CPU utilization is always more than the source utilization. Considering the CPUs are idle at the beginning of the transfer, and homogeneous systems are used as source and destination, it is acceptable to use the destination CPU usage for the flow throughput. The output of the algorithm is a set of sampling throughput values derived by using the suggested number of nodes, number of stripes per node and the number of parallel streams per stripe. The capacity of a flow and the number of streams used for that capacity are decided by the optimal throughput calculated by Newton's model(Lines 2-3) based on the parallelism levels selected by the selection algorithm(Algorithm 2). Before increasing the sampling stripe number, we set sampling parallel stream number per stripe to the sampling number before the optimal stream number and the throughput for that stream number is set for the single stripe value(Lines 4-5). This value corresponds to the X_{fk} variable in the flow

model. The X_{ij} value depends on the arc. If it is a network or disk system arc, that corresponds to the throughput of total number of stripes, otherwise it is the throughput of stripes per node.

If the capacity of the flow is reached the NIC capacity, then stripe value per node is set as 1 for each node, because further increasing the striping in the same node will not give any improvement for the throughput. In this case, the node number is exponentially increased with the current stream and stripe number, until the network capacity is reached and the throughput starts to drop down, or the available nodes are all consumed(Lines 6-16). The algorithm returns the sampling throughput values for the stripes and number of nodes, number of stripes per node and number of stream per stripe.

If the card limit is not reached, that means there is still room for increasing the stripe number in the same node. However, CPU and disk capacity or the network could present a bottleneck. First, the remaining capacity of CPU and NIC is calculated by subtracting the throughput of the current striping level (Line 18). The CPU utilization is transformed into the throughput unit by using the regression model presented in Section 4.3.1. The throughput curve usually shows the characteristics of the logarithmic increase in our experiments. So, the difference of the throughput of the next sampling interval can not be greater than twice the previous interval, as we increase the sampling number exponentially by powers of two. We calculate this difference to decide if there is any room for an additional stripe in the same node(Line 19). While this value is less than the left capacity of CPU and NIC the striping value is exponentially increased by 2(Lines 20-29). However this loop can stop if the throughput level starts to decrease indicating that the capacity of the network or the disk is reached(Lines 24-26). If the network and the disk limit is not reached but the node capacity is exceeded(Line 30), then the current stripe and stream value for the node is set, the node number is increased exponentially by using this stripe and stream values until that capacity or the available resource number is reached(Lines 31-39). Finally the node number, stripe number and the stream number is returned with their throughput values for further evaluation.

Figures 5.1,5.2 and 5.3 present the algorithm output for different experimental cases to better describe the working logic behind. In Figure 5.1, the transfers occur between a set of 16 nodes with 1G interfaces. The sampling algorithm stops at 16 nodes, 1 stripe per node and 2 streams per stripe.

Algorithm 3 Optimization Algorithm(OPT_B)

Input: Th_N: a set of throughput values for different parallelism levels from the sampling algorithm and their corresponding CPU utilization values, U_{NIC}, N_{avail}, $U_{CPU_{dest}}$

Output: Th_S: a set of throughput values for different node, stripe, streams values, number of parallel streams per stripe(N_k), number of stripes per node (S_x), number of nodes(N_n)

$j \leftarrow 1$ $S_x \leftarrow 1$ $N_n \leftarrow 1$ $Limit \leftarrow 0$
$U_f \leftarrow$ Optimal throughput decided by the Newton model based on Th_N
$N_{opt} \leftarrow$ Optimal stream number decided by the Newton model based on Th_N
$N_{Si} \leftarrow$ Sampling parallel stream number before N_{opt}
$Th_{Sj} \leftarrow$ Striping throughput for current $S_x \leftarrow$Th$_{N_{Si}}$
if $U_f \approx U_{NIC}$ **then**
    while $Limit = 0$ && $2 * N_n <= N_{avail}$ **do**
        $N_n \leftarrow N_n \times 2$
        $Th_{Sj+1} \leftarrow$ getThroughput(N_n,S_x,N_{Si})
        if $Th_{Sj+1} < Th_{Sj}$ **then**
            $Limit \leftarrow 1$
        end
        $j \leftarrow j + 1$
    end
    return Th_S, N_n, S_x,N_{Si}
end
else if $U_f < U_{NIC}$ **then**
    $U_{CPU_{left}} \leftarrow U_{CPU} - Th_{N_{Si}}$, $U_{NIC_{left}} \leftarrow U_{NIC} - Th_{N_{Si}}$, $U_{left} \leftarrow min(U_{CPU_{left}}, U_{NIC_{left}})$
    $PI \leftarrow$ Improvement in throughput between the current and the previous sampling level $\leftarrow$Th$_{NSi} -$ Th_{NSi-1}
    while $2 \times PI < U_{left}$ **do**
        $S_x \leftarrow S_x \times 2$
        $Th_{Sj+1} \leftarrow getThroughput(N_n, S_x, N_{Si})$
        $PI \leftarrow Th_{Sj+1} - Th_{Sj}$
        if $Th_{Sj+1} < Th_{Sj}$ **then**
            $Limit \leftarrow 1$ Break the loop
        end
        $U_{CPU_{left}} \leftarrow U_{CPU} - Th_{Sj+1}$, $U_{NIC_{left}} \leftarrow U_{NIC} - Th_{Sj+1}$, $U_{left} \leftarrow min(U_{CPU_{left}}, U_{NIC_{left}})$
        $j \leftarrow j + 1$
    end
    if $Limit = 0$ **then**
        while $2 * N_n <= N_{avail}$ **do**
            $N_n \leftarrow N_n \times 2$
            $Th_{Sj} \leftarrow$ getThroughput(N_n,S_x,N_{Si})
            if $Th_{Sj} < Th_{Sj-1}$ **then**
                $Limit \leftarrow 1$ Break the loop
            end
            $j \leftarrow j + 1$
        end
    end
    return T_S, N_n, S_x, N_{Si}
end

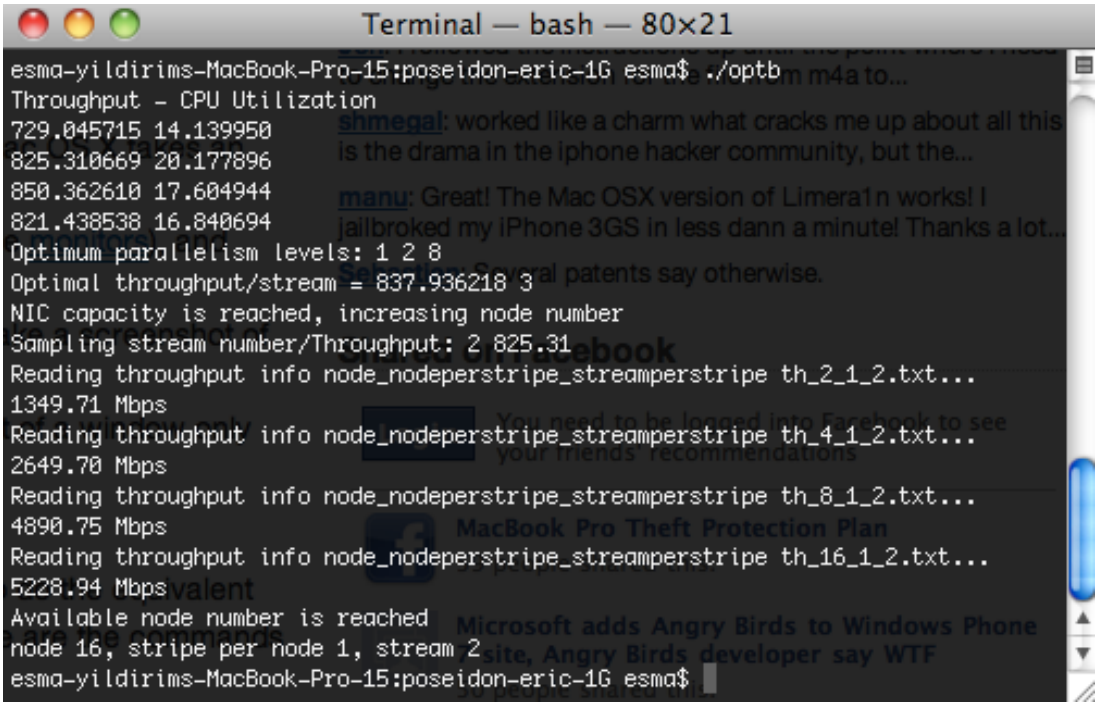


FIGURE 5.1: Algorithm output(OPT_B) for Figure 6.3

In this case the NIC capacity is reached after 1 stripe and the node number is increased exponentially. In Figure 5.2, 2 nodes with 10G interfaces are available. The CPU capacity is reached after 4 stripes and 2 stream per stripe. The node number is increased from this point on and it stops at 2 since the available node number is reached. Finally in Figure 5.3, we see a case where 16 nodes with 1G cards are available, however sampling stops after 8 nodes since the network capacity is reached.

5.2 Optimization Algorithm for Unknown Disk and Network Capacity in Heterogeneous Resources(OPT_H)

The OPT_B algorithm assumes both source and destination nodes are homogeneous. However in cases where the source and destination clusters are of different architecture, it is important that both source and destination capacities are taken into account. For example, the cpu model and the number as well as the architecture or the interconnects could be different on both ends. In this algorithm(Algorithm 4), we alter the OPT_B algorithm so that these issues are taken into account.

Input parameters needed by the algorithm are a set of throughput values from the sampling algorithm(Th_N) along with their CPU utilization values, the source and destination NIC capacities ($U_{NIC_{src}}$,$U_{NIC_{dest}}$), available node numbers ($N_{avail_{src}}$,$N_{avail_{dest}}$) and CPU capacities ($U_{CPU_{src}}$,

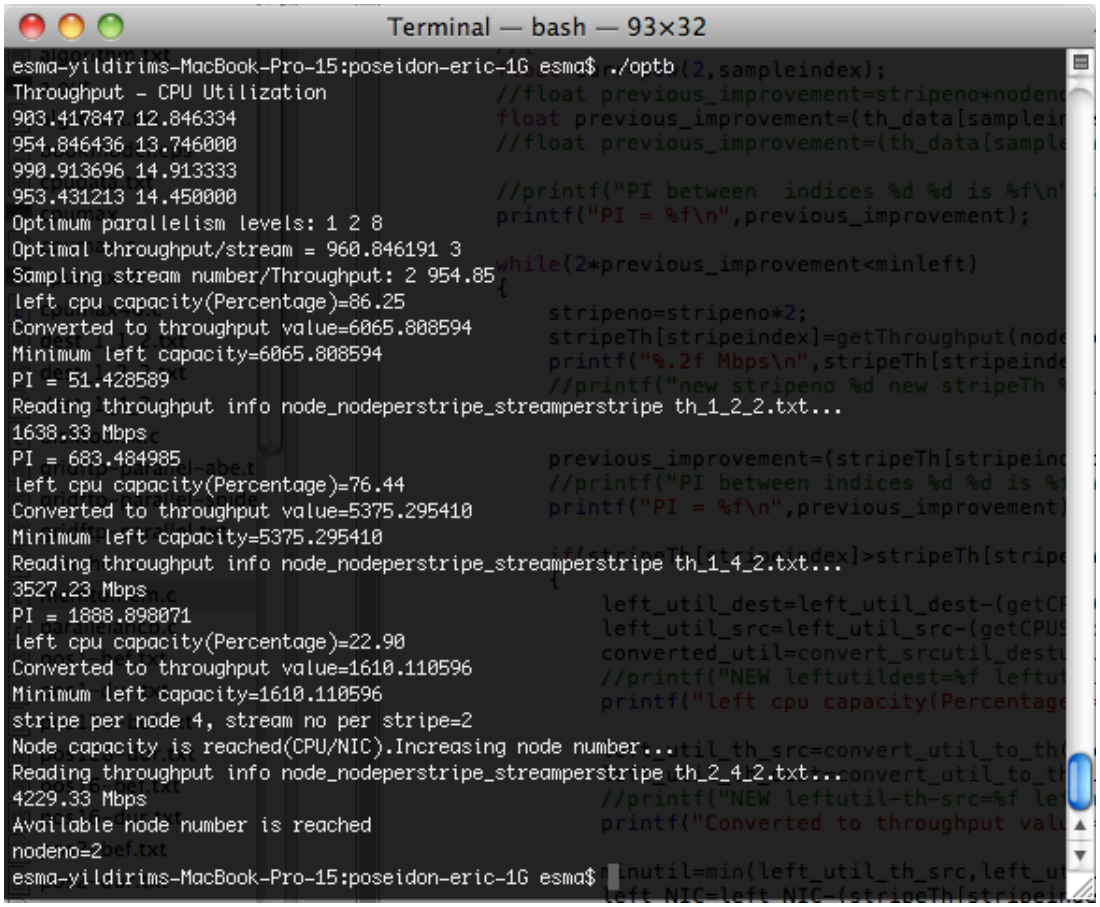


FIGURE 5.2: Algorithm output(OPT_B) for Figure 6.1

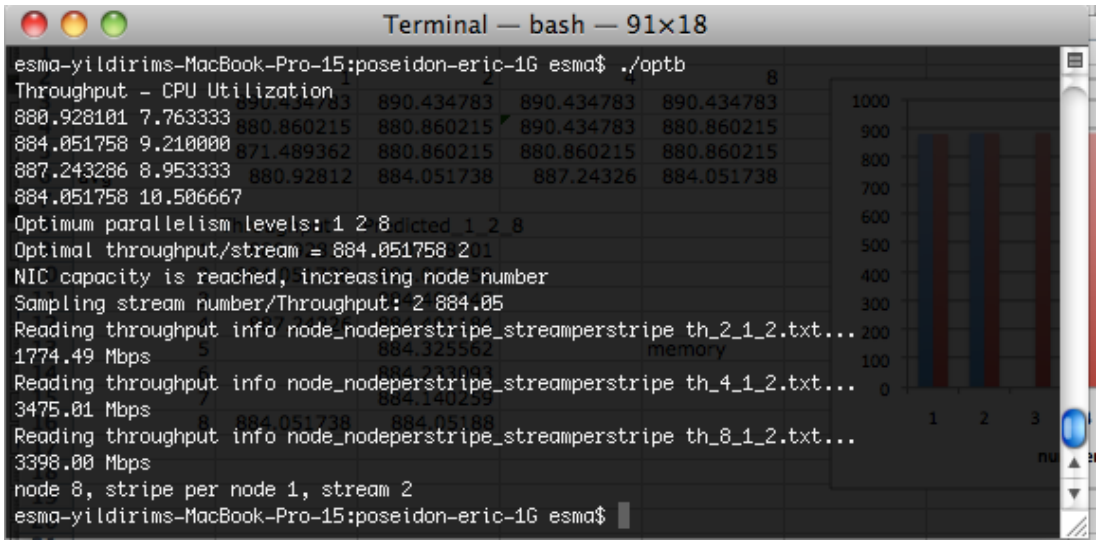


FIGURE 5.3: Algorithm output(OPT_B) for Figure 6.12

$U_{CPU_{dest}}$). The output of the algorithm is a set of throughput values along with its stream per stripe, source and destination stripe per node numbers and the number of source and destination nodes. Lines 42-46 show similarities with the beginning of the OPT_B algorithm. The minimum values of NIC and CPU capacities are calculated for both source and destination. If the optimal throughput has already reached the maximal capacity of the source node(Line 49), the left capacity of the destination node is calculated(Lines 50-51) as well as the PI value. If there is still room for further increase in the striping value in the destination node and also the source available node number is not reached, then the stripe number for the destination is increased along with the node number for the source exponentially(Lines 53-54). The throughput value for the current stream, stripe and node values is read(either by sampling or from history). If the current throughput value is less than the previous throughput value, then either the disk or the network has reached its limit(Lines 56-58). In this case the loop is stopped, otherwise the new left capacity of the destination node is calculated(Lines 59-60).

If the destination node has reached its capacity, the same steps from Line 50-61 is repeated but this time by exchanging the source and destination for the variables. If none of the nodes has reached its capacity, that means we can increase the stripe number on the same source and destination node until either of the nodes reaches its capacity (Lines 68-80).

In the continuation of the algorithm, both the source and destination nodes reach their capacity and the increase of the node number is inevitable as long as the available node numbers are not exceeded(Algorithm 5).

We present a sample output for the OPT_H algorithm in Figure 5.4. In this test case, the source and destination NICs are heterogeneous and transfers occur in wide-area. The source NIC capacity is reached after 2 streams and the destination CPU capacity is reached after 4 stripes. The available node number is reached after 8 stripes in total and a throughput of 5.8Gbps is reached.

Algorithm 4 Optimization Algorithm(OPT_H)

Input: Th_N, $U_{NIC_{src}}$, $U_{NIC_{dest}}$, $N_{avail_{src}}$, $N_{avail_{dest}}$, $U_{CPU_{src}}$, $U_{CPU_{dest}}$

Output: Th_S, N_k, $S_{x_{src}}$,$S_{x_{dest}}$, $N_{n_{src}}$,$N_{n_{dest}}$

42 $j \leftarrow j+1$ $S_{x_{src}} \leftarrow 1$ $S_{x_{dest}} \leftarrow 1$ $N_{n_{src}} \leftarrow 1$ $N_{n_{dest}} \leftarrow 1$ $Limit \leftarrow 0$

43 $U_f \leftarrow$ Optimal throughput decided by the Newton model based on Th_N

44 $N_{opt} \leftarrow$ Optimal stream number decided by the Newton model based on Th_N

45 $N_{Si} \leftarrow$ Sampling parallel stream number before N_{opt}

46 $Th_{Sj} \leftarrow Th_{N_{Si}}$

47 $U_{min_{src}} \leftarrow min(U_{CPU_{src}}, U_{NIC_{src}})$

48 $U_{min_{dest}} \leftarrow min(U_{CPU_{dest}}, U_{NIC_{dest}})$

49 **if** $U_f \approx U_{min_{src}}$ **then**

50 $\quad U_{CPU_{left_{dest}}} \leftarrow U_{CPU_{dest}} - Th_{S_{Ni}}$, $U_{NIC_{left_{dest}}} \leftarrow U_{NIC_{dest}} - Th_{S_{Ni}}$

51 $\quad U_{min_{left_{dest}}} \leftarrow min(\mathrm{U}_{CPU_{left_{dest}}}, U_{NIC_{left_{dest}}})$

52 $\quad PI \leftarrow Th_{NSi} - Th_{NSi-1}$

53 $\quad$ **while** $2 \times PI < U_{min_{left_{dest}}}$ && $N_{n_{src}} \times 2 <= N_{avail_{src}}$ **do**

54 $\quad\quad S_{x_{dest}} \leftarrow S_{x_{dest}} \times 2$, $N_{n_{src}} \leftarrow N_{n_{src}} \times 2$

55 $\quad\quad Th_{Sj+1} \leftarrow$ getThroughput($N_{n_{src}}$,$S_{x_{src}}$,$N_{n_{dest}}$,$S_{x_{dest}}$,N_{Si})

56 $\quad\quad$ **if** $Th_{Sj+1} < Th_{Sj}$ **then**

57 $\quad\quad\quad Limit \leftarrow 1$ Break the loop

58 $\quad\quad$ **end**

59 $\quad\quad U_{CPU_{left_{dest}}} \leftarrow U_{CPU_{dest}} - Th_{S_{j+1}}$, $U_{NIC_{left_{dest}}} \leftarrow U_{NIC_{dest}} - Th_{S_{j+1}}$

60 $\quad\quad U_{min_{left_{dest}}} \leftarrow min(\mathrm{U}_{CPU_{left_{dest}}}, \mathrm{U}_{NIC_{left_{dest}}})$

61 $\quad\quad j \leftarrow j+1$

62 $\quad$ **end**

63 **else if** $U_f \approx U_{min_{dest}}$ **then**

64 $\quad$ Repeat Lines 50-61 exchanging *src* and *dest* labels

65 **else if** $U_f < min(U_{min_{src}}, U_{min_{dest}})$ **then**

66 $\quad$ Repeat Lines 50-51 for both *src* and *dest* values

67 $\quad PI \leftarrow Th_{NSi} - Th_{NSi-1}$

68 $\quad$ **while** $2 \times PI < U_{min_{left_{dest}}}$ || $2 \times PI < U_{min_{left_{src}}}$ **do**

69 $\quad\quad$ **if** $2 \times PI < U_{min_{left_{dest}}}$ && $2 \times PI < U_{min_{left_{src}}}$ **then**

70 $\quad\quad\quad S_{x_{src}} \leftarrow S_{x_{src}} \times 2$, $S_{x_{dest}} \leftarrow S_{x_{dest}} \times 2$

71 $\quad\quad\quad$ Repeat Lines 55-58

72 $\quad\quad\quad$ Repeat Lines 59-60 for both *src* and *dest* values

73 $\quad\quad\quad j \leftarrow j+1$

74 $\quad\quad$ **else if** $2 \times PI < U_{min_{left_{dest}}}$ && $N_{n_{src}} \times 2 <= N_{avail_{src}}$ **then**

75 $\quad\quad\quad$ Repeat Lines 54-61;

76 $\quad\quad$ **else if** $2 \times PI < U_{min_{left_{src}}}$ && $N_{n_{dest}} \times 2 <= N_{avail_{dest}}$ **then**

77 $\quad\quad\quad$ Repeat Line 75 by exchanging *src* and *dest* labels

78 $\quad\quad$ **else**

79 $\quad\quad\quad$ Break the loop

80 $\quad\quad$ **end**

81 $\quad$ **end**

82 **end**

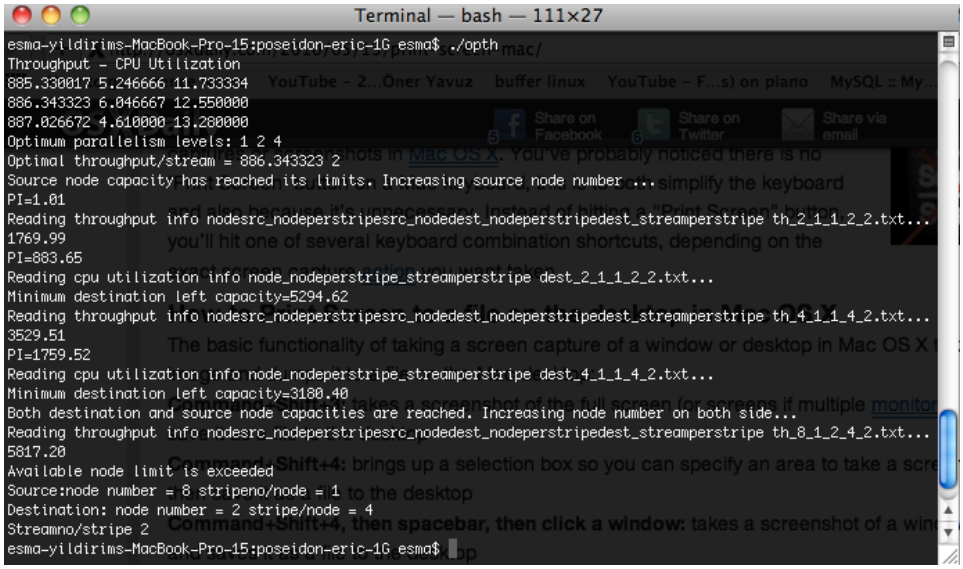


FIGURE 5.4: Algorithm output(OPT_H) for Figure 6.21

Algorithm 5 Continuation of Optimization Algorithm(OPT_H)

if $Limit = 0$ **then**
  while $N_{n_{src}} \times 2 <= N_{avail_{src}}$ && $N_{n_{dest}} \times 2 <= N_{avail_{dest}}$ **do**
  end
  $N_{n_{src}} \leftarrow N_{n_{src}} \times 2$
  $N_{n_{dest}} \leftarrow N_{n_{dest}} \times 2$
  $Th_{Sj+1} \leftarrow$ getThroughput($N_{n_{src}}$,$S_{x_{src}}$,$N_{n_{dest}}$,$S_{x_{dest}}$,N_{Si})
  if $Th_{Sj+1} < Th_{Sj}$ **then**
    Break the loop
  end
end
return $T_S, N_{n_{src}}, N_{n_{dest}}, S_{x_{src}}, S_{x_{dest}}, N_{Si}$

Chapter 6
Experimental Results

The experiments designed to show how the optimization algorithms work are conducted over the LONI and the TeraGrid network by using clusters that have 1G and 10G NIC network cards on their nodes. Table 6.1 presents the list of resources used for the experiments.The 10G nodes over the LONI network are limited to two nodes. The nodes with 1G network cards are allocated through the PBS batch job scheduler. Hence the number of nodes that are allocated is also limited. We made use of MPI to be able to make multiple requests to different GridFTP servers run on user mode over the allocated nodes. Each node on Eric, Oliver and Poseidon clusters has 4 processor cores except for the head node which has 8 cores. Abe cluster GridFTP nodes also have 4 cores in each. Lincoln cluster has 8 cores in each node.

The experiments are divided into sections based on the transfer path(e.g. disk-to-disk), the algorithm used and the type of network(e.g. WAN).

6.1 Local Area Experiments for OPT_B Algoritm(LONI Network)

In this section, we present the experimental results regarding the local area transfers conducted over the LONI network and algorithm application for both disk-to-disk and memory-to-memory transfer cases.

TABLE 6.1: Testbed

Resource	Network Interface	Available node number
eric.loni.org	10G	2
eric.loni.org	1G	128
oliver.loni.org	10G	2
oliver.loni.org	1G	128
poseidon.loni.org	10G	2
poseidon.loni.org	1G	128
gridftp-abe.ncsa.teragrid.org	10G	2
lincoln.ncsa.uiuc.edu	1G	256

6.1.1 Disk-to-disk Transfers

The application of the optimization algorithm(OPT_B) for disk-to-disk transfers are presented in this section. To better utilize the parallel file system speed, multi file parallel writes are done which are explained in Section 3.4 for each stripe. Figure 6.1 shows the experimental results between Oliver and Eric cluster nodes with 10G interfaces and 4 cores on each. The throughput values of a set of different parallelism levels that are decided by the sampling algorithm and their corresponding cpu utilization values are presented in Figure 6.1.a and b. The light blue bars in Figure a. shows the throughput values fed into the OPT_B algorithm. The optimal stream value predicted by the Newton's model is 3 using the parallelism levels of 1, 2 and 8 decided by the selection algorithm. The cpu utilization for that value is around 15%. As the throughput gained by the optimal stream number reaches neither NIC limit nor the CPU capacity, the algorithm set the stream number to the sampling stream number before the optimal which is 2 and starts increasing stripe number exponentially. In Figure 6.1.c we could see the data points used by the algorithm with light blue bars. The horizontal axis values represent the node number, total number of stripes and stream number per stripe in order. The total utilization for the node passes 80% at 4 stripes(Figure 6.1.d) and since the left capacity is not enough for the next improvement in throughput the node number is increased to 2 by setting the stripe per node to 4. Our experiments also showed that if we did not increase the node number but used the same node only by increasing the stripes it caused the throughput fall and all the cpu cores hit their maximum capacity. The algorithm stops at 2 nodes because the maximum available node with 10G cards is 2. However if further nodes were available it would continue to increase the node number until the throughput falls. The Newton model is applied to the available data points by using the stripe values 1,4 and 8. The prediction gives very accurate results for the actual throughput values. The maximum throughput reached is 4.2 Gbps. Considering the disk maximum capacity is around 6 Gbps and it is shared among all of the LONI users, it is a very high value gained with limited resources. Also from the possible stream, stripe and node value of $8 \times 8 \times 2 = 128$ combinations only 7 data points are needed.

We applied the algorithm also by using the head node of the Eric and Poseidon clusters which have 8 cores. In Figure 6.2.a the prediction model suggests the optimal stream number as 4 based on

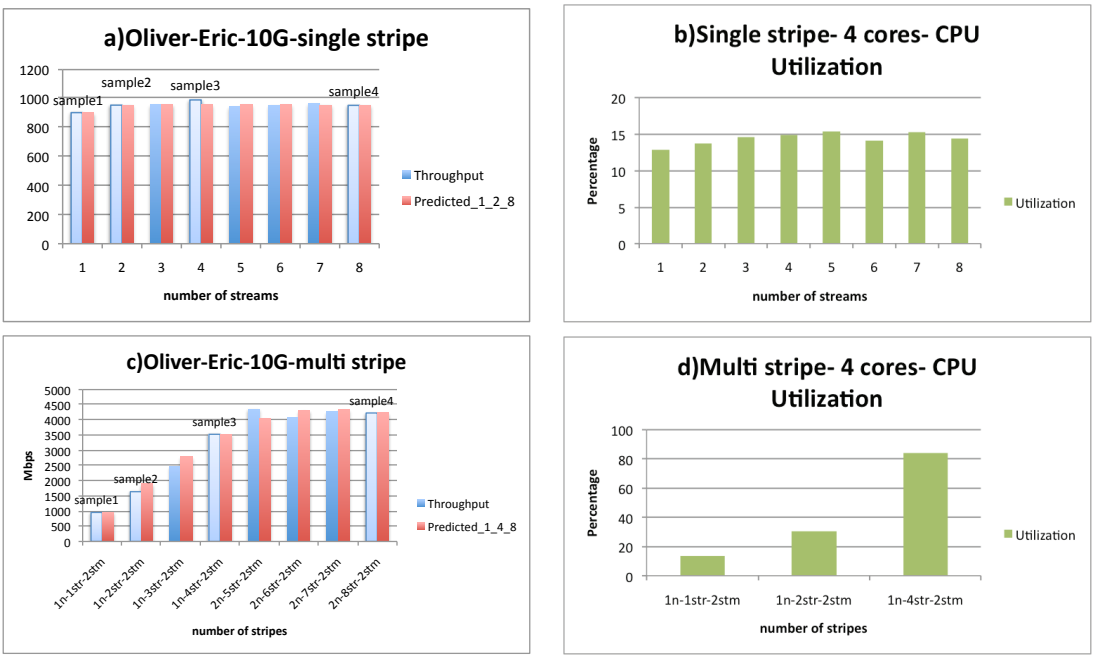


FIGURE 6.1: Model Application on Disk-to-Disk Transfers between Eric and Oliver with 10G interfaces and 4 cores

parallelism levels 1,2 and 8. The cpu utilization for that number is around 7%(Figure 6.2.b) which is almost half of the utilization with the previous results for 4 cores. The algorithm fixes the parallel stream number and increases the stripe number exponentially until either the cpu or NIC capacity is reached(Figure 6.2.c). The stripe number is increased until 8 which indicates in Figure 6.2.d reaches the cpu capacity to almost 100%. The stripe number for the node is set to 8. Unfortunately only one node is available with 8-cores hence the node number can not be increased. The application of Newton's model on the results with parallelism levels 2,4 and 8 suggests 8 as the optimal stripe number. The maximal throughput achieved is around 3.8 Gbps. From $8 \times 8 \times 1 = 64$ possible combinations only 7 data points are needed.

In the last experiment, we allocate 16 nodes with 1G interfaces and apply the algorithm for transfers between Eric and Oliver clusters. In Figure 6.3.a and b, we present the throughput values of different parallel stream values that are fed into the algorithm along with their cpu utilization values. The model predicts 3 as the optimal stream number. Considering the throughput is around %90 of NIC capacity the stripe number is set to one and the stream number for stripe is set to the sampling stream number before the optimal which is 2. In Figure 6.3.c the light blue bars shows the data

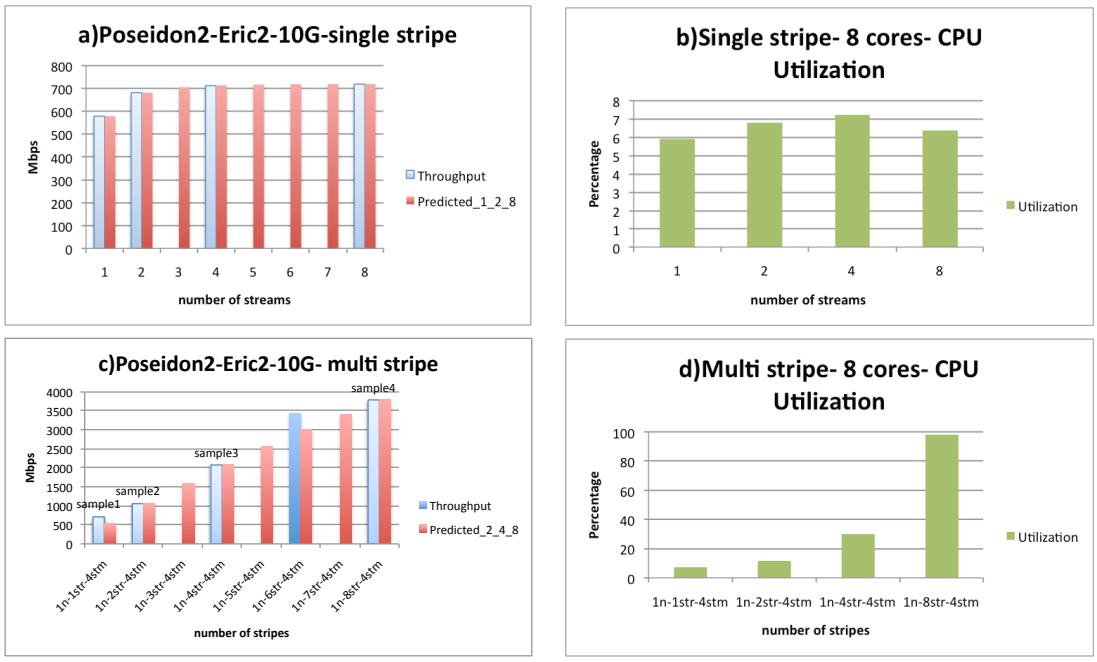
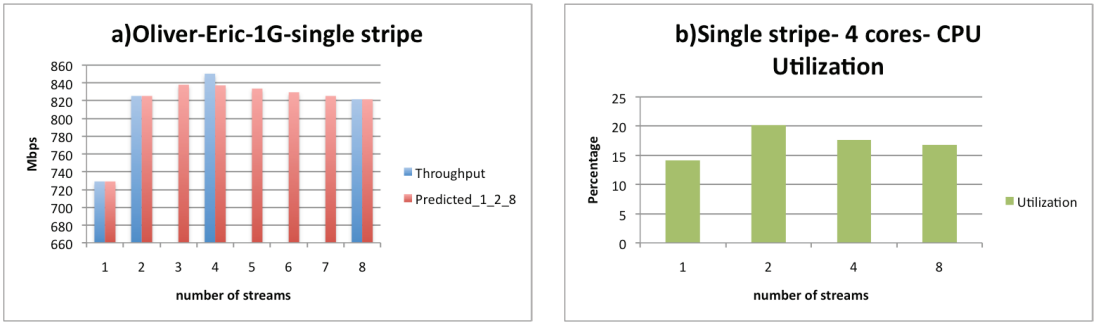


FIGURE 6.2: Model Application on Disk-to-Disk Transfers between Eric and Poseidon head nodes with 10G interfaces and 8 cores



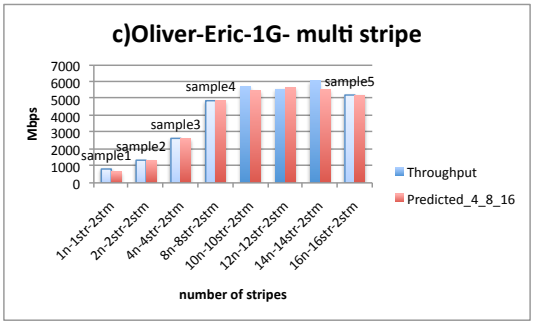


FIGURE 6.3: Model Application on Disk-to-Disk Transfers between Eric and Oliver with 1G interfaces

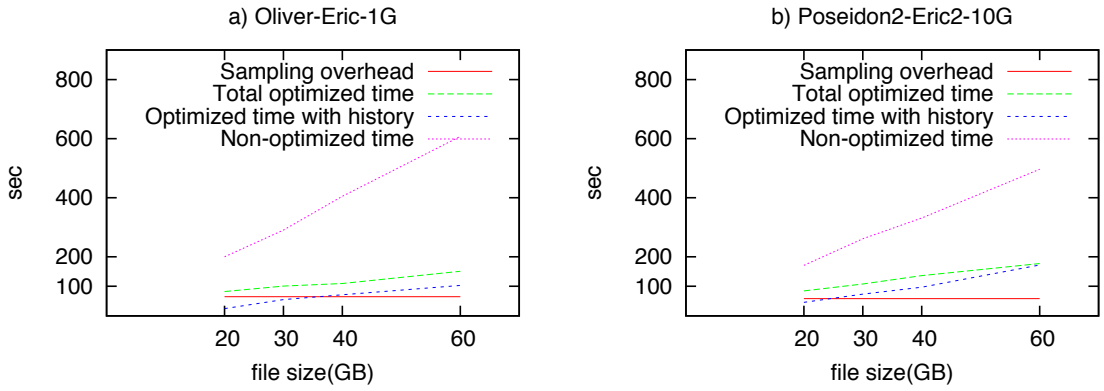


FIGURE 6.4: Transfer times for model application with 1GB sampling size vs data size

points used by the algorithm. The node number is increased until 16 which is the available node number. Based on the striping levels 4,8 and 16, the prediction model provides accurate results and presents the optimal striping number as 12. The maximal throughput gained is around 5.8Gbps which is very close to the maximal disk speed. From $8 \times 1 \times 16 = 128$ possible combinations only 8 data points are needed by the algorithm.

Another set of experiments are run to compare the transfer times for different data sizes between 20GB-60GB by using 1GB sampling size to analyze the overhead of the model. The experiments are conducted by using both 1G and 10G interfaces. Figure 6.4.a presents the results for 1G interface over Oliver and Eric clusters. Sampling overhead is the total time to do the samplings needed by the algorithm each of which is 1GB. Total optimized time is the overhead time + the time needed to transfer the rest of the file with the optimized parameters. The optimized time with history represents the time to transfer the whole file with the previous optimized parameters. The non-optimized time is the transfer time with 1 stream only. For 20GB data size, the optimized total time is decreased by half even with sampling overhead comparing to non-optimized transfer time. This ratio is increased by 5 for the transfers with history information. The gap between the optimized and non-optimized time is increased as the file size increases. Similar results are observed with the 10G interface experiments(Figure 6.4.b). This proves that the model provides very high throughput results even with the sampling overhead included.

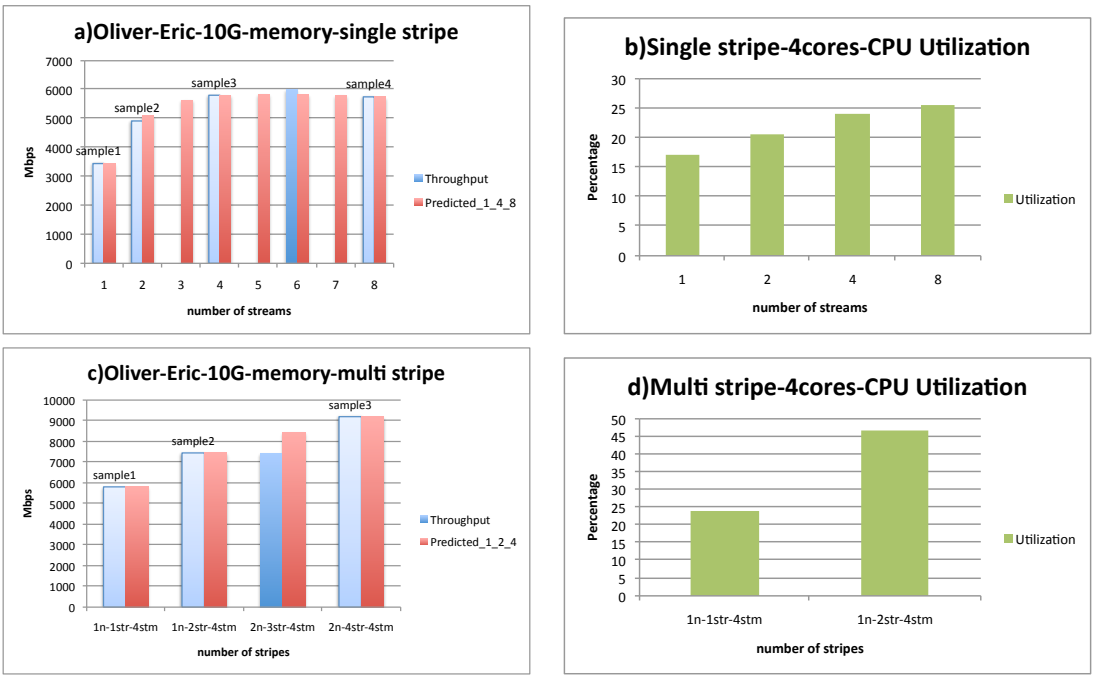


FIGURE 6.5: Model Application on Memory-to-memory Transfers between Eric and Oliver with 10G interfaces and 4 cores

6.1.2 Memory-to-memory Transfers

In this section, we present the results of similar tests for algorithm application in the previous subsection but using memory-to-memory transfers. The transfers are done using the Eric and Oliver clusters with 1G and 10G interfaces.

In Figure 6.5.a the light blue bars indicate the set of throughput values given from the sampling algorithm to the OPT_B algorithm. The optimal stream number is 6 however neither CPU (Figure 6.5.b) nor the NIC capacity is reached. Hence the sampling number for a single flow is selected as 4 (which is the previous sampling before optimal stream number) and the stripe number is increased from this point on. When the stripe number is 2 the throughput reaches a 7.5Gbps but the NIC capacity does not have the available room for the next improvement step. Therefore the stripe per node number is fixed at 2 and the node number is increased to 2. The throughput achieved reaches 9Gbps which is around the maximum value for the switch and the theoretical network capacity. The algorithm stops at this point because the available node number is reached. The total number of data values is 8 from the minimum possible combination of 8x2x2=32 values.

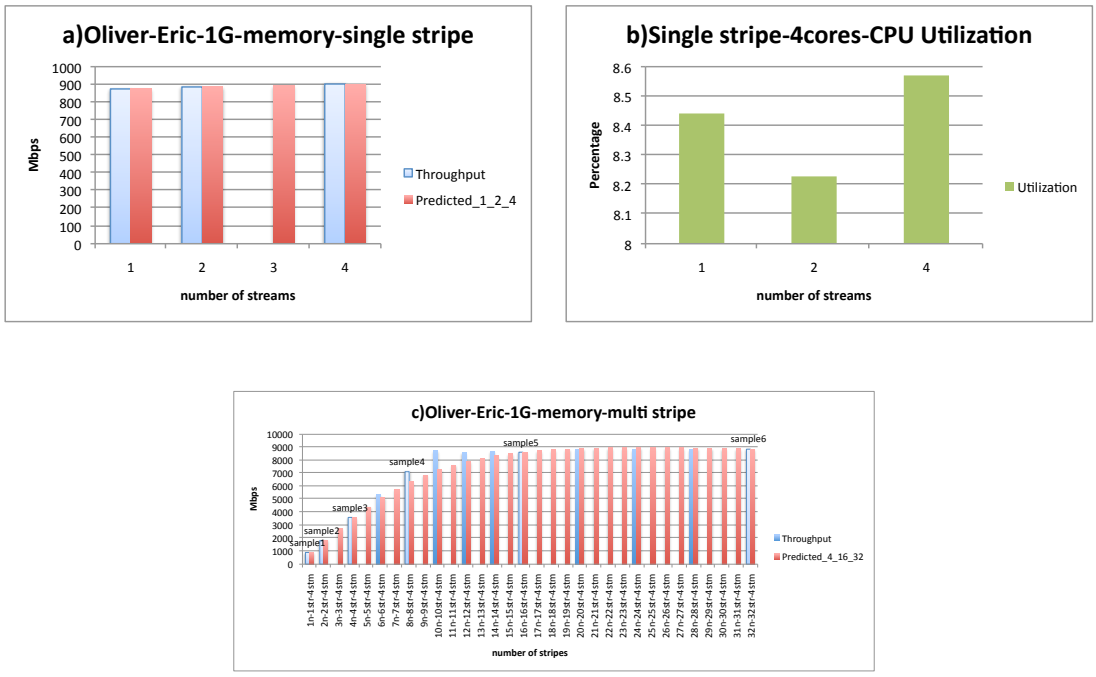


FIGURE 6.6: Model Application on Memory-to-memory Transfers between Eric and Oliver with 1G interfaces

In the second set of experiments, 1G interfaces are used. In Figure 6.6.a, the optimal stream number is 4 and the NIC capacity is reached with this throughput value. Hence the stripe per node is also set as 1 and the node number is increased exponentially(Figure 6.6). The maximum throughput is reached around 10 streams and the prediction curve gives a value around 16 streams which also gives maximal throughput. 9 data points are used among 4x32=128 minimum possible combinations.

Finally, the overhead of the sampling data is calculated to see the effectiveness of the model with 1GB and 2GB sampling sizes and a data size range of 20-60GB is used. Figure 6.7.a presents the transfer times using 1GB sampling size. There is a big gap between the total optimized time which includes the sampling overhead and non-optimized times and this difference gets bigger when history transfers are used. When we increase the sampling size to 2GB(Figure 6.7.b), this overhead of sampling increases but then again with the optimization improvement, the transfer time is twice less than the non-optimized version for 20GB data size. With history samplings the improvement is very high. When we look at the 10G interface results(Figure 6.7.c and d), the gap between the optimized and non-optimized transfers is less because this time the NIC does not bring a bottleneck to the non-optimized transfers. Then again the total optimized time provides half the time of non-optimized

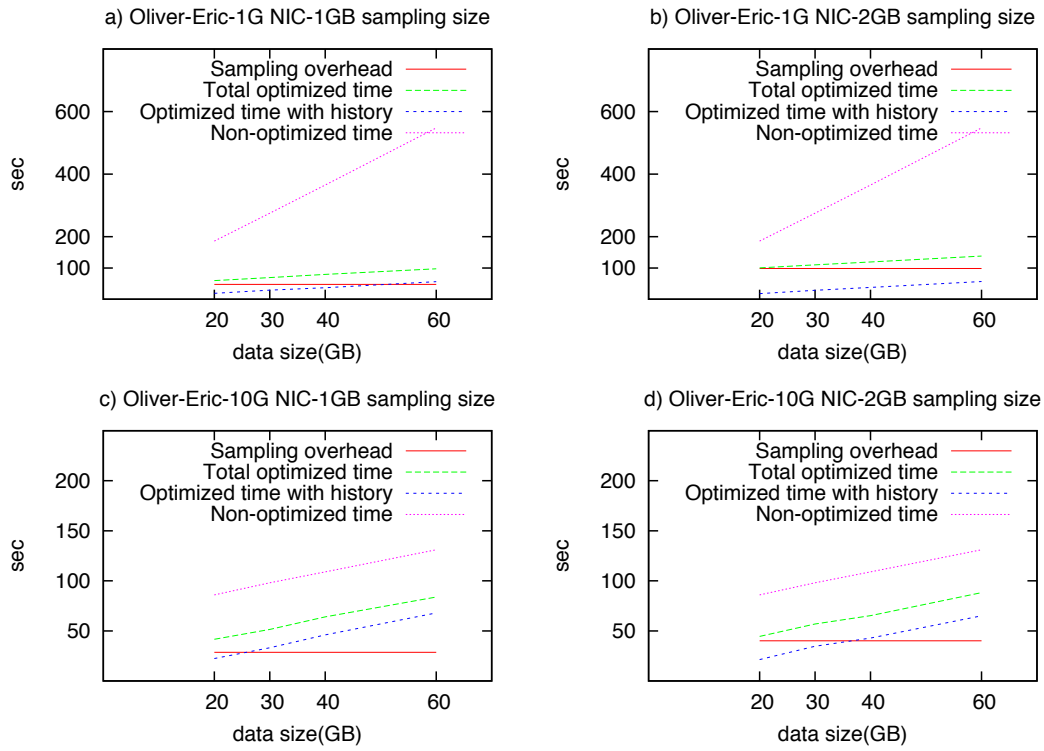


FIGURE 6.7: Transfer times for model application with 1GB and 2GB sampling size vs data size in memory-to-memory transfers

transfers. With the history prediction information, this difference is much bigger due to the lack of sampling overhead.

6.2 Wide-area Experiments for OPT_B Algoritm (LONI -TeraGrid Network)

The wide area transfers are conducted between two sites; LONI and NCSA. For 10G interface tests, the GridFTP nodes of the Abe cluster and the 10G nodes of Eric cluster is used. For 1G interface another cluster called Lincoln is used on the NCSA site. The destination for these transfers are nodes with 1G interfaces of Eric and Poseidon clusters on LONI site.

6.2.1 Disk-to-disk Transfers

In this section, we present the results of two different experiments that include disk-to-disk transfers with 10G and 1G interfaces. Figure 6.8 shows the results with 10G interfaces. In Figure 6.8.a the total sampling information that is fed to the OPT_B algorithm is 4 and the maximum parallel stream number used is 8. The Newton's model predicts the optimal stream number as 4 which is also

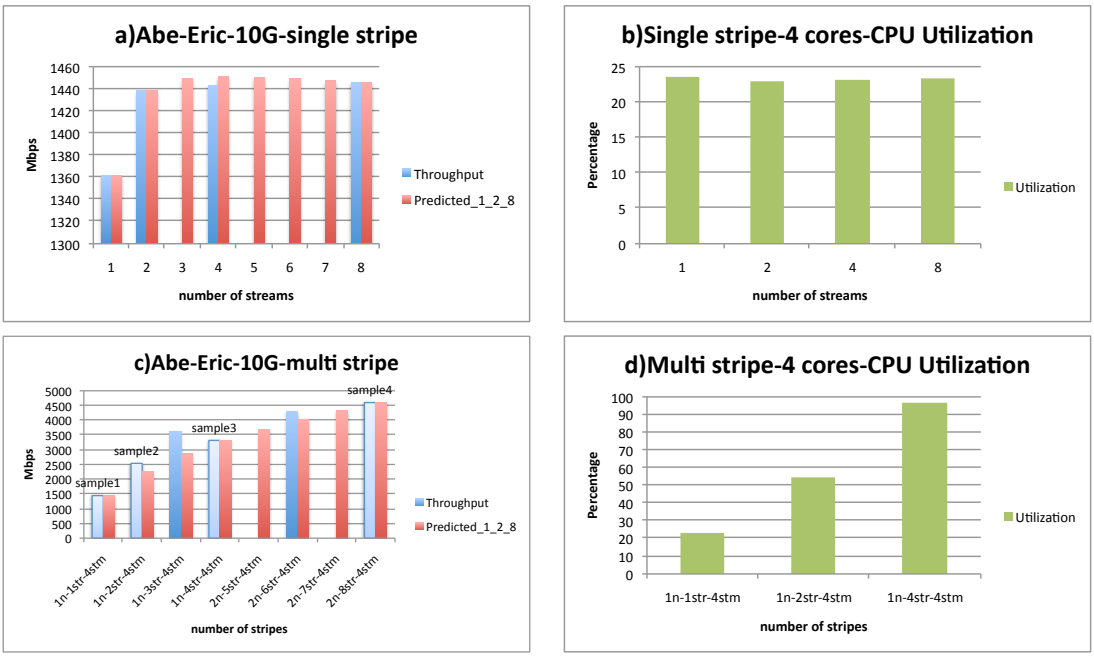


FIGURE 6.8: Model Application on Disk-to-disk Transfers between gridftp-abe and eric nodes with 10G interfaces and 4 cores

the sampling number before the optimal stream number. The CPU utilization is around 24% for a single stripe(Figure 6.8.b). Setting the stream number per stripe to 4, the stripe number is increased exponentially(Figure 6.8.c). The light blue bars presents the sampling levels chosen by the algorithm. However after 4 stripes per node the CPU utilization reaches %90 which does not leave space for the next improvement value(Figure 6.8.d). Therefore the stripe per node is set to 4 and the node number is increased. The available node number is reached with 8 stripes hence the algorithm stops at this point. The prediction applied based on these stripe-node-stream values suggests 8 as the optimal number. From a possible minimum combination of 8x8x2=128 values only 7 is selected for the model and a throughput value of 4.5Gbps is achieved.

In Figure 6.9, we see the application of the OPT_B algorithm on disk-to-disk transfers conducted over the 16 allocated nodes of Lincoln and Poseidon clusters with 1G network interfaces. The optimal stream number predicted by the Newton's iteration is 8 and the throughput is around 900Mbps which indicates the network card limit is reached. After that point, the number of nodes is increased exponentially(Figure 6.9.c). It stops at 16 nodes since the available node number is reached and the throughput does not fall down. 3Gbps throughput is reached and 8 sampling values are used.

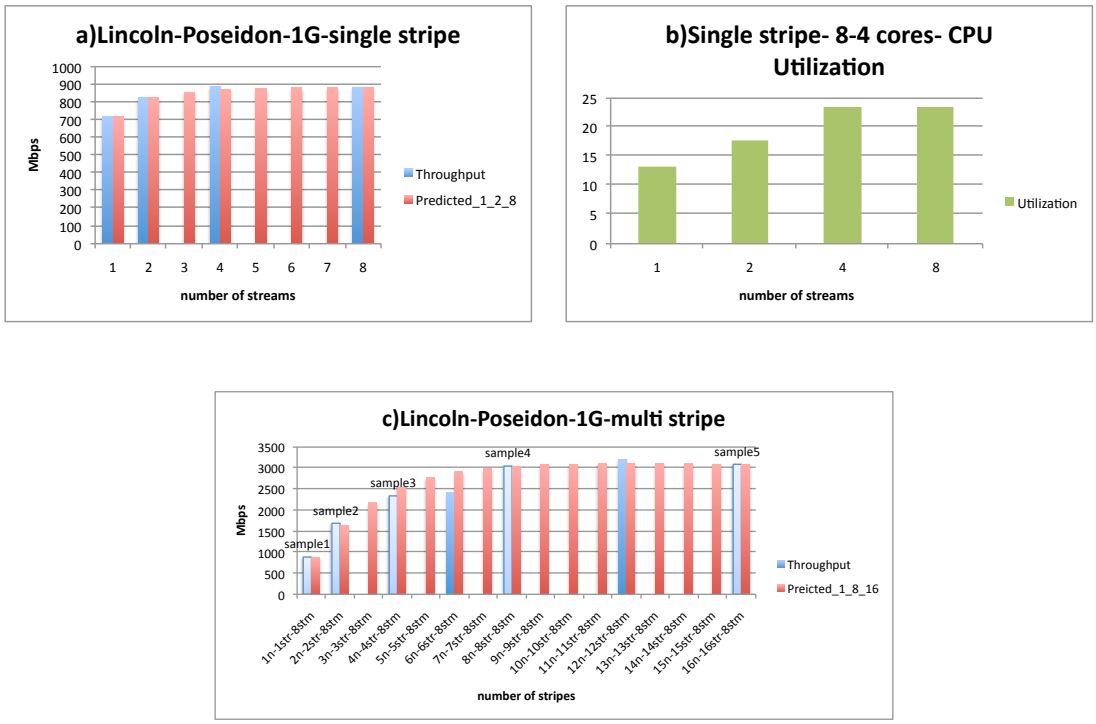


FIGURE 6.9: Model Application on Disk-to-disk Transfers between Lincoln and Poseidon clusters with 1G interfaces

The overhead of the OPT_B algorithm for the disk-to-disk transfers between LONI and Teragrid sites with 10G and 1G network interfaces is presented in Figure 6.10. The results show that in both cases, our model gives better results and the gap between the optimized and non optimized times gets bigger as the data size increases. The improvement is much more significant in case of the transfers with history prediction.

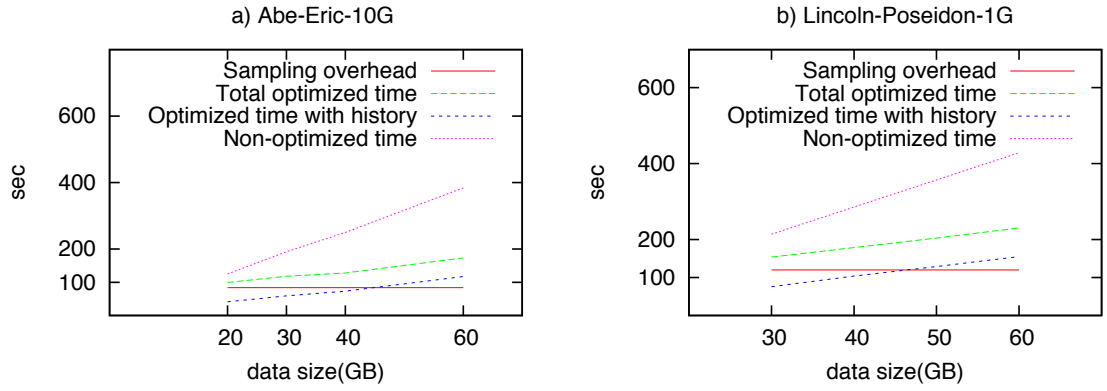


FIGURE 6.10: Transfer times for model application with 2GB sampling size vs data size

6.2.2 Memory-to-memory Transfers

In this section, we present the results of OPT_B algorithm application for memory-to-memory transfers. The 10G interface tests are conducted between the headnode of Eric which has 8 cores and gridftp-abe that uses two nodes with 4 cores in round-robin fashion. In this way, the total number of cores are the same for both source and destination. In Figure 6.11.a we see the throughput of single stripe by using parallel streams. The blue bars shows the parallelism levels fed into the algorithm as input. The highest throughput is gained around 5-6 streams which is 4Gbps. However when we look into the average CPU utilization, the highest value is around %13 which is around %100 for one core(Figure 6.11.b). But the NIC limit is not reached yet. The CPU becomes the bottleneck in this case. The sampling stream number before the optimal is 4. Setting this number for each stripe, the stripe number is increased exponentially. The light blue bars in Figure 6.11.c represents the striping levels used by the algorithm. The average CPU utilization increases to %37 but never passes this point(Figure 6.11.d) but the throughput starts to decrease after 8 stripes on the same node although the NIC and CPU limits are not reached. In this case network becomes the bottleneck. The algorithm stops after 16 stripes on the same node. The suggested node number is 1 and the stripe per node is 6. From 1x16x16=256 possible minimum number of combinations, only 9 is used by the OPT_B algorithm and a throughput of 6Gbps is achieved.

Figure 6.12 presents the results for 1G interface between Lincoln and Eric clusters. In Figure 6.12.a, the throughput results are very close to each other and the NIC capacity is reached. The optimal stream selected by the Newton's model is 2. The CPU utilization is around %10. In this case the stripe number is set to 1 for a single node and the node number is increased exponentially(6.12.c). The throughput starts to decrease after 4 nodes and the algorithm stops at 8 nodes. The predicted optimal stripe number is 6 and although the predicted optimal throughput is a little less than the actual throughput the striping level selected is very accurate. A total of 8x1x8=64 possible node-stripe-stream combinations, only 7 points are used. The total throughput achieved is more than 5Gbps.

The overhead of the algorithm application for both cases is presented in Figure 6.13. The sampling sizes for the experiments are 1GB and 2GB. The total optimized time for both sampling sizes are

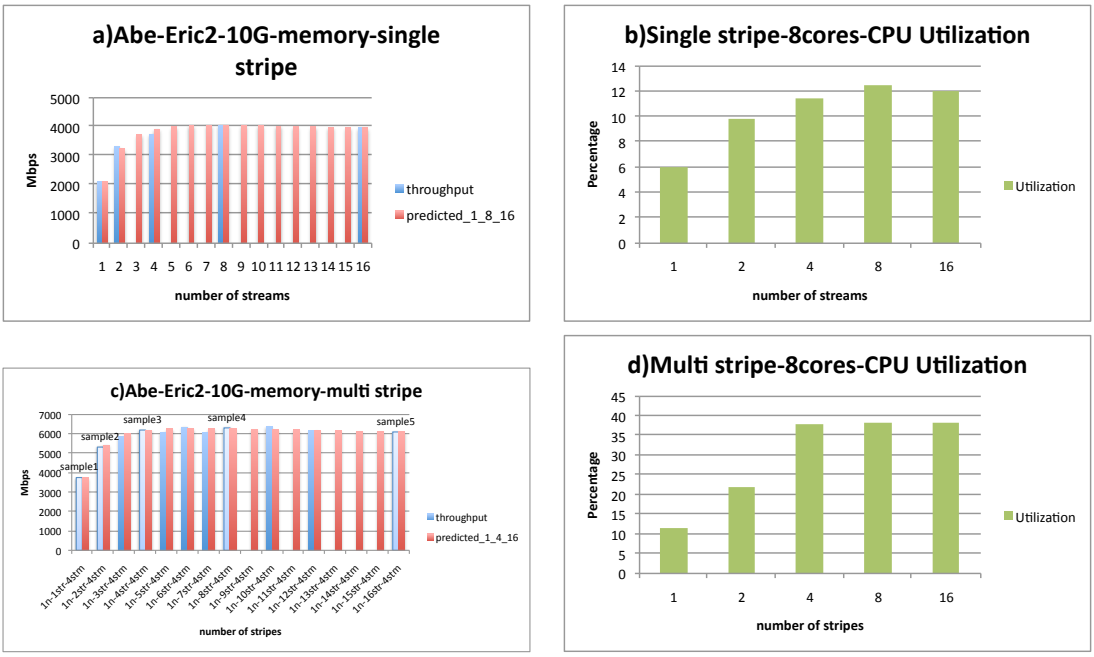


FIGURE 6.11: Model Application on Memory-to-memory Transfers between gridftp-abe and Eric2 with 10G interfaces and 2 4-core nodes on source and 1 8-core node on destination

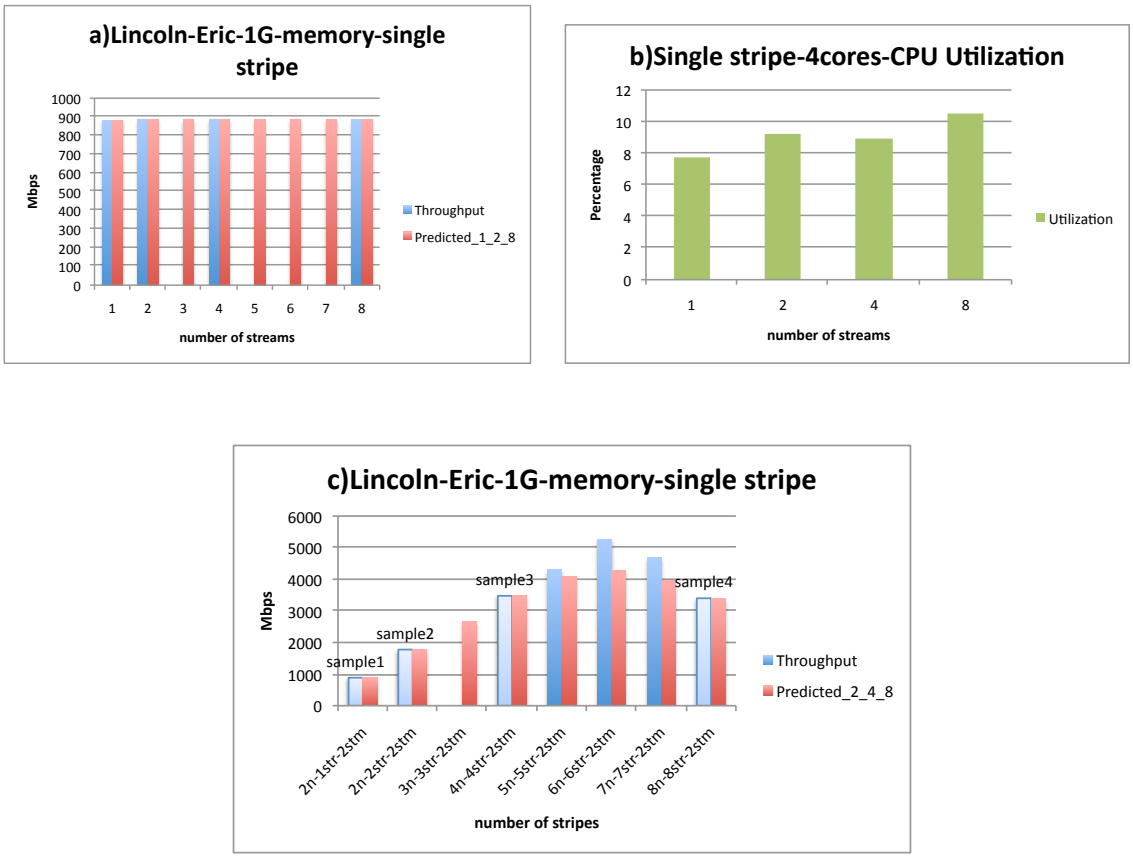


FIGURE 6.12: Model Application on Memory-to-memory Transfers between Lincoln and Eric with 1G interfaces

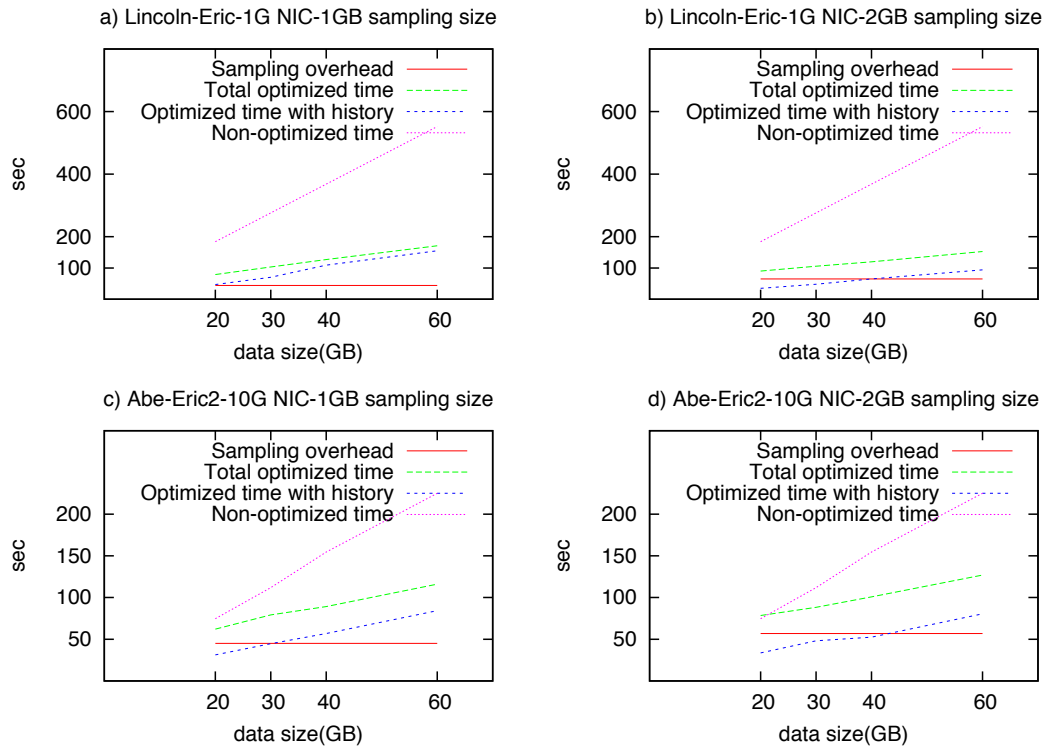


FIGURE 6.13: Transfer times for model application with 1GB and 2GB sampling size vs data size in memory-to-memory transfers between NCSA and LONI

similar. However if we look into the history results the 2GB sampling gives a better accuracy since the time to transfer is far less then the result with 1GB sampling size. However as the data size increases 2GB sampling performs better than 1GB sampling. For both cases, the distinction between the non-optimized and optimized transfers are very high which proves the efficiency of our model. For the 10G interface case the gap between the non-optimized and optimized times is less for small data size however it increases as the data size increases. 1GB sampling performs better than 2GB sampling since the 2GB samplings only increases the overhead time providing the same accuracy. However the optimization performs better in all of the cases except for 2GB sampling-20GB data size case.

6.3 Local Area Experiments for OPT_H Algoritm(LONI Network)

The experiments designed for the OPT_H Algorithm takes into consideration the network interface card differences at the source and destination as well as the processor architecture.

6.3.1 Disk-to-disk Transfers

In this test case, we consider different processor architectures. The cards on both destination and source sides are 10Gbps. The source node Oliver2 has 4 dual core 2.33GHz processors which correspond to 8 cores in total. The destination Eric1 has 2 dual core 3GHz processors which also correspond to 4 cores. During the test the architecture of Eric2 is assumed to be the same as Eric1. However the measurements that will decide the node number increase point are done on Eric1. In Figure 6.14.a we see that the parallel stream number does not affect the throughput much. The destination and source CPU utilizations are in the range % 10-16(Figure 6.14.b) although the destination utilization is much bigger. The optimal stream number set by the algorithm is 2 and the stripe number is increased(Figure 6.14.c). The x-axis labels are ordered for source node number, total source stripe number,destination node number, total destination stripe number and the stream number per stripe. When the stripe number is reached at 4, the destination CPU capacity does not have idle time enough for the next improvement(Figure 6.14.d). Hence the node number on the destination is increased while the source node number stays the same. However since the available destination node number is 2, the algorithm stops at this point. The total number of samplings needed is 8 out of a possible minimum 8x8x8x2x1=1024 combinations. 4.5Gbps throughput is achieved.

6.3.2 Memory-to-memory Transfers

The same test case for the heterogeneous processor architectures in the previous section is repeated in this section for memory-to-memory transfers. Since the disk bottleneck is eliminated the effect of the parallel streams is more significant (Figure 6.15.a). The gap between source and destination utilization for parallel streams is higher in Figure 6.15.b. The stream number set by the algorithm in this case is 8. The stripe number is increased at both source and destination since the CPU and NIC capacity is not reached. However after 2 stripes the throughput reaches around 7.7Gbps and since there is only a single source available, and the NIC capacity do not have enough room for further increasing the stripe number, the algorithm stops at this point. Since only 2 points are available and the cause of the termination is the insufficient node number, it is wise to use the latest sampling throughput as the optimal throughput. Even if we increased the stripe number in the same node the

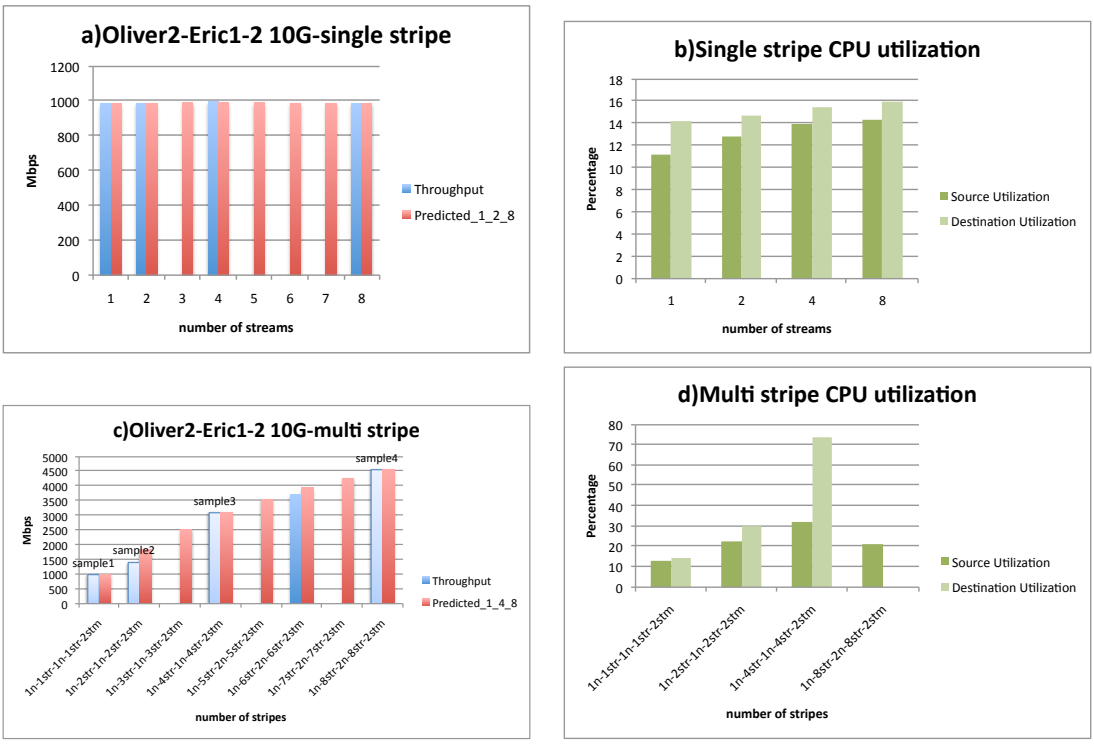


FIGURE 6.14: Model Application (OPT_H) on Disk-to-disk Transfers between Oliver2 and Eric1&2

achievable throughput only reaches 8 Gbps. In this case 6 samplings out of a possible 16 combinations are used and 7.7 Gbps throughput is achieved.

The second test case analyzes the heterogeneity of NICs on source and destination for memory-to-memory transfers. We allocated 16 nodes with 1G cards from the Poseidon cluster as source, and we used 2 head nodes of Eric cluster with 10G cards. When we look at the effect of parallel streams on the end-to-end throughput, we can see that it is very slight and the throughput is limited by the source NIC bottleneck(Figure 6.16.a). Since the CPU utilizations show no bottleneck, the algorithm sets the parallel stream number to 2 and increases the source node number exponentially(Figure 6.16.c). When we reach 4 stripes, the destination utilization reaches a point where there is not enough idle CPU cyles for the next improvement value (Figure 6.16.d). In this case, the destination node number is also increased. However since the available destination node number is reached at this point, the algorithm stops here. The achievable throughput is 6.2Gbps and a total of 6 samples are needed from a possible combination of 4x8x1x2x4=256 values.

In Figure 6.17, we present the transfer times of the model application with OPT_H algorithm with 2GB samplings. The results of the transfers using different node architectures indicate intermingled

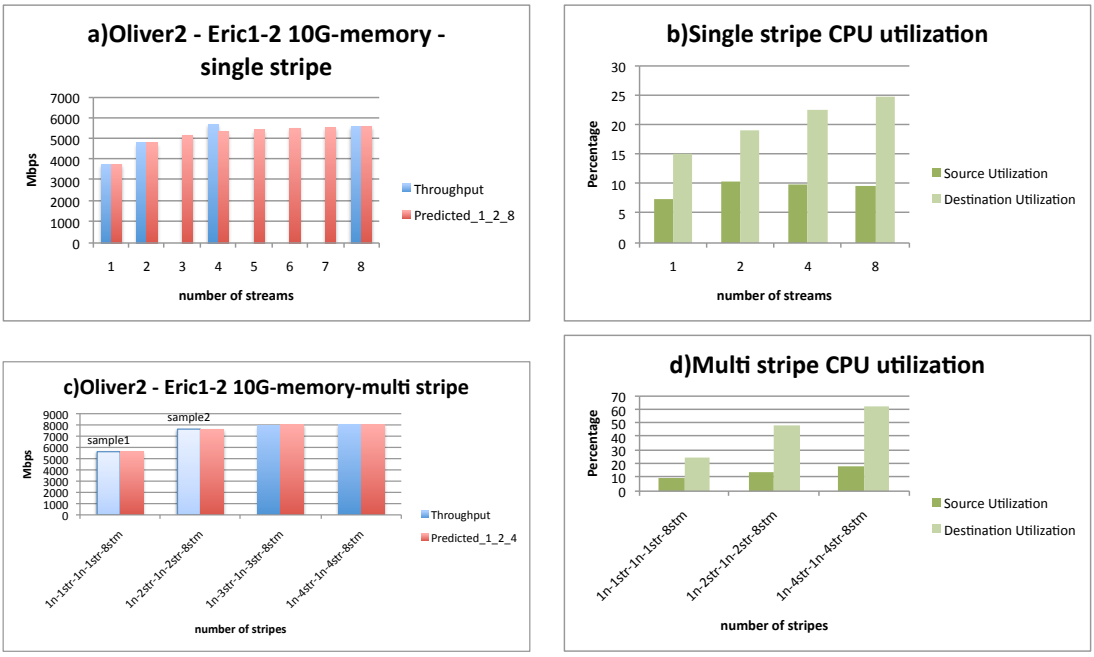


FIGURE 6.15: Model Application (OPT_H) on Memory-to-memory Transfers between Oliver2 and Eric1&2

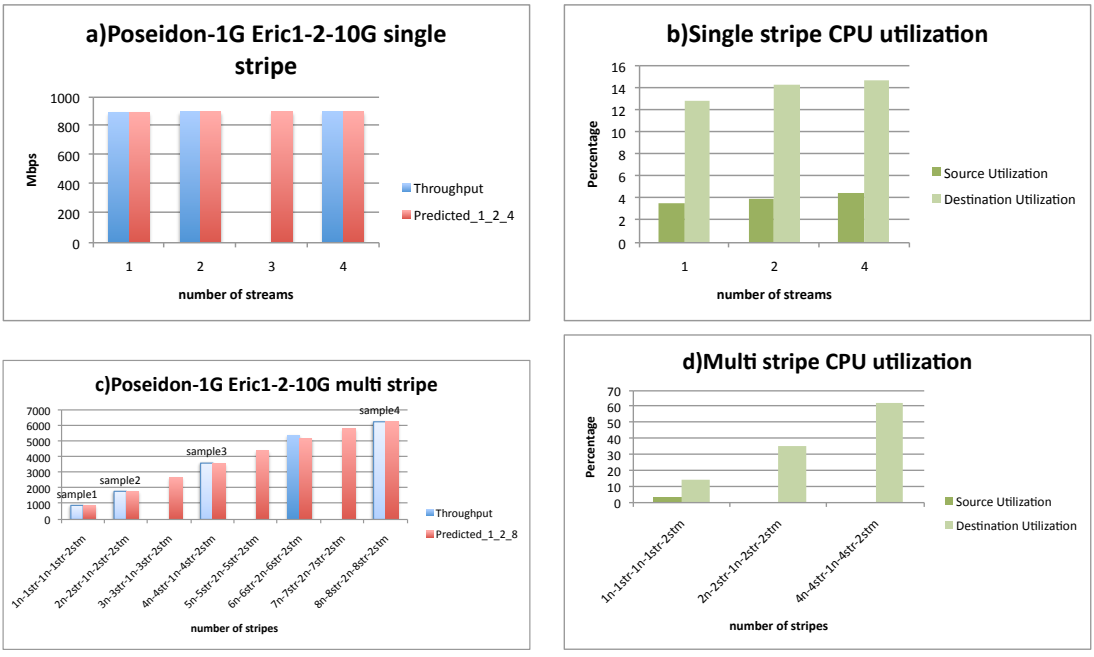


FIGURE 6.16: Model Application (OPT_H) on Memory-to-memory Transfers between Poseidon(1G) and Eric1&2(10G)

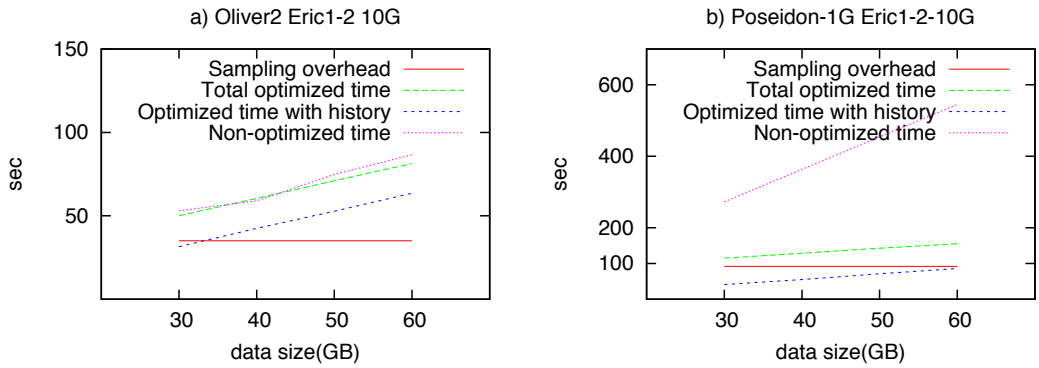


FIGURE 6.17: Transfer times for model application(OPT_H) with 2GB sampling size vs data size in memory-to-memory transfers in LONI

lines between non-optimized and total optimized time(Figure 6.17.a). This is due to the large sampling size overhead and the lack of available node numbers to properly apply the algorithm. However the optimized time with history prediction gives good results. The transfers that focus on the heterogeneity of NICs on the source and destination nodes provide much better improvements in terms of total transfer times(Figure 6.17.b). The optimized time with history is 9 times better than the non-optimized time for the 30GB data size.

6.4 Wide-area Experiments for OPT_H Algoritm (LONI -TeraGrid Network)

In this section, we present the experimental results of transfers between NCSA and LONI and algorithm application for both disk-to-disk and memory-to-memory transfer cases.

6.4.1 Disk-to-disk Transfers

The differences in the network interfaces is an important issue that can affect the end-to-end throughput. In this test case, we measure the effectiveness of the OPT_H algorithm where the interfaces on the source and destination are different. We allocated 16 nodes from the Lincoln cluster with 1G network interfaces which resides in NCSA. As the destination nodes, we used the 2 head nodes of Eric cluster (LONI) which both have 10G interfaces. In Figure 6.18.a the throughput reaches 880Mbps which indicates that the source node interface capacity is reached. Setting the stream number per stripe to 2, the node number in the source is increased. After 8 stripes, the destination capacity is also reached and the node number in the destination is also increased along with the source(Figure

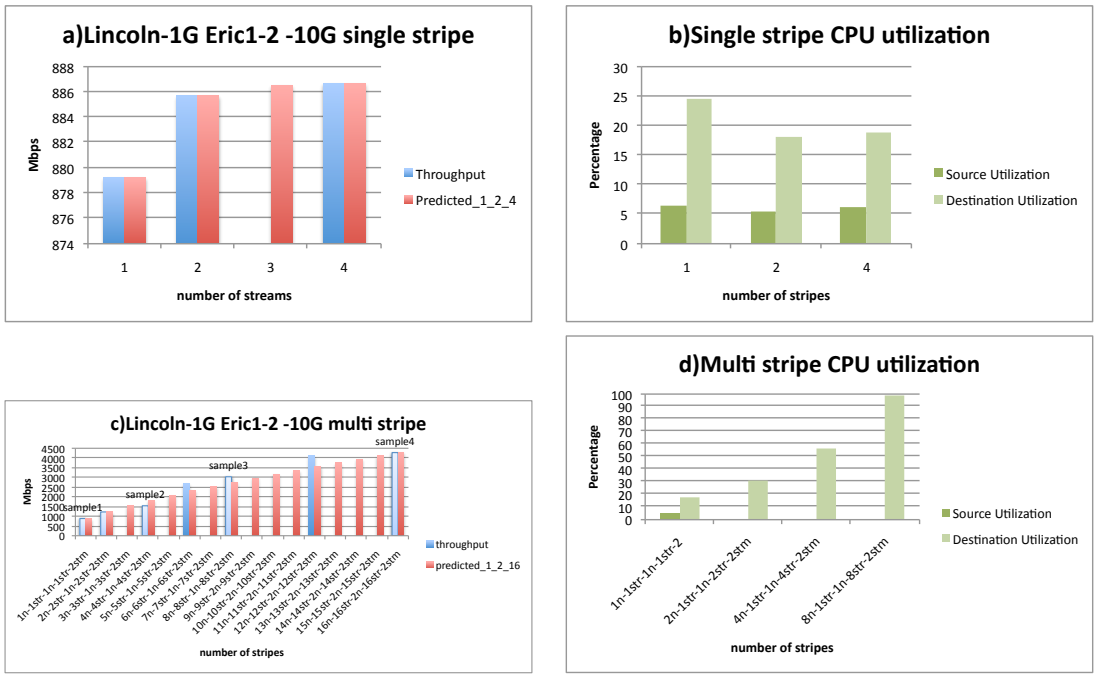


FIGURE 6.18: Model Application (OPT_H) on Disk-to-disk Transfers between Lincoln(1G) and Eric(10G)

6.18.c). Since at this point the available node number in both sides is reached the algorithm stops. A disk-to-disk throughput of 4Gbps is achieved and 7 samplings are used from a possible combination of 4x16x1x8x2=256 values.

In the second test case, we applied the model on disk-to-disk transfers of different processor architectures between 2 4-core 2.8GHz processor gridftp nodes of Abe cluster in NCSA site and Eric2 which is an 8-core 2.33GHz head node of the Eric cluster in LONI site. In Figure 6.19.a, we see the effect of the parallel streams on the disk throughput. The optimized stream number selected by the algorithm is 4. By setting the stream/stripe number to that value, the stripe number is increased(Figure 6.19.c). However when the stripe number is 8, the source utilization reaches its capacity(Figure 6.19.d). In this case, the source node number is increased by the OPT_H algorithm to 2.But the throughput decrease at 16 stripe. 9 data points are used by the algorithm out of a possible 16x2x8x1x16=4096 possible combinations. The achieved throughput is around 2.9Gbps. The low value could be related to either disk load or the variation of the CPU utilization on the destination node which gives us a low average utilization value although the capacity is reached.

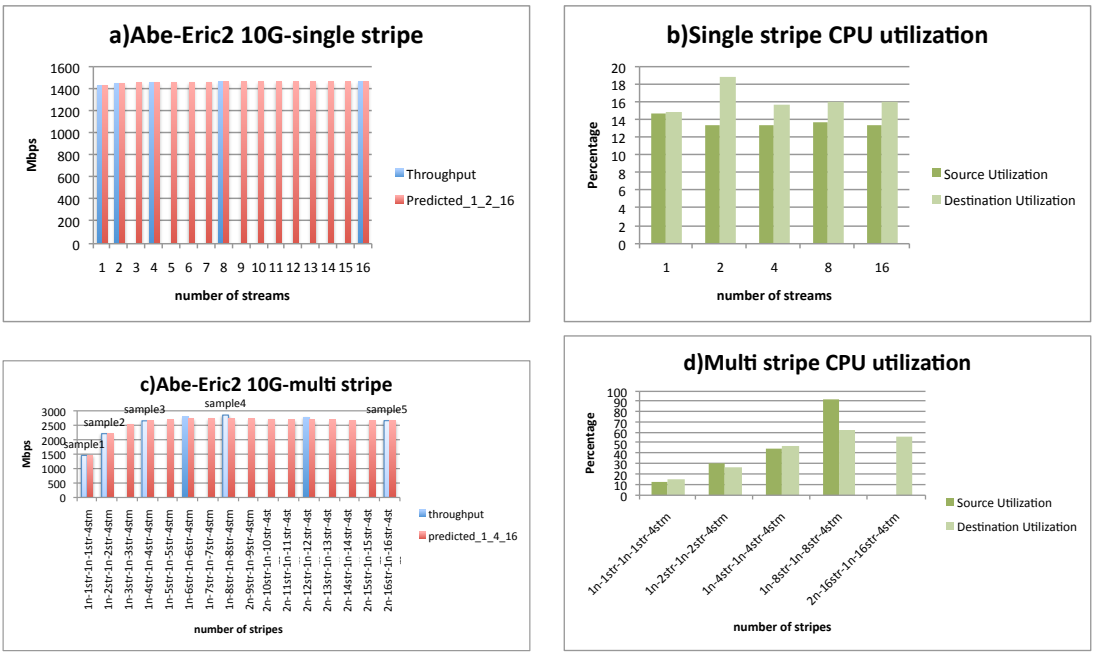


FIGURE 6.19: Model Application (OPT_H) on Disk-to-disk Transfers between Abe(10G) and Eric2(10G)

In Figure 6.20, we present the disk-to-disk transfer times of wide area experiments with 2GB sampling size and compare the optimized time of OPT_H algorithm to the non-optimized time. The results in Figure 6.20.a shows us the improved transfer times in case of the heterogeneous network interface cards in source and destination. The gap between the non-optimized and optimized times is very large due to the 1G NIC bottleneck at the source side. The test results of the different processor architectures also shows an improvement in the transfer time especially for the optimized time with history information(Figure 6.20.b).

6.4.2 Memory-to-memory Transfers

The same tests about the network interface differences at source and destination in the previous section is repeated for memory-to-memory transfers. In Figure 6.21.a, we see a similar characteristics comparing to the disk transfers in terms of parallel streams except that the CPU utilizations are less(Figure 6.21.b). The optimal stream number is set to 2 by the algorithm and the stripe number is increased by increasing the node number at the source(Figure 6.21.c). However after 4 stripes, the destination node cpu utilization reaches a level where further increase in the same node is not feasible. In this case the destination node number is increased to 2 which is also the available node number for

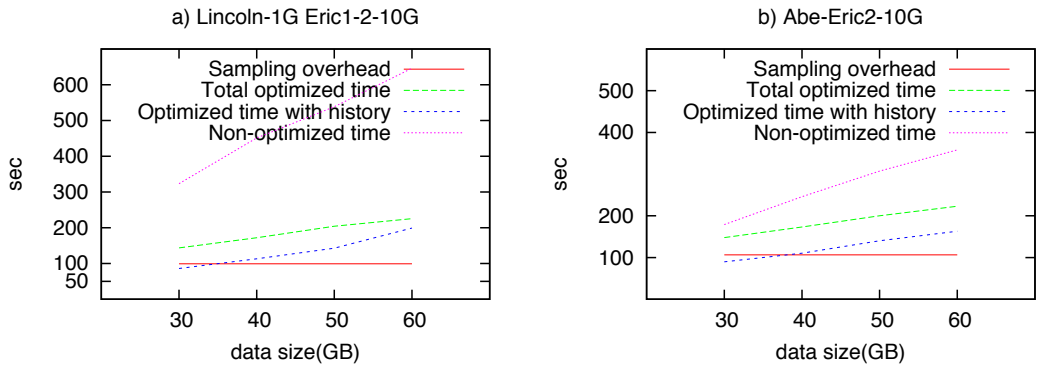


FIGURE 6.20: Transfer times for model application(OPT_H) with 2GB sampling size vs data size in disk-to-disk transfers between NCSA and LONI

the destination. Hence, the algorithm stops at this point giving a throughput value around 6Gbps. A total of 6 sampling values are needed to apply the algorithm.

The second test case is also a similar test case for the different processor architectures presented in the previous section, however we use memory-to-memory transfers. Figure 6.22.a indicates that the throughput reaches 4Gbps around 4-8 streams while the source utilization is around %25 which corresponds to %100 for single core. One important observation in this case is that the source utilization(4 cores) is greater than the destination utilization(8cores). Setting the stream number per stripe to 4 streams, the stripe number is increased exponentially. This continues until 16 stripe where the throughput drops down. But the end-systems capacities are not reached indicating the network limit is reached in this case. A throughput 5Gbps is achieved and 9 data points are used among a possible minimum 16x1x16=256 combinations.

The overhead of the application of the OPT_H algorithm for memory-to-memory transfers is presented in Figure 6.23. The gap between the non-optimized and optimized transfers (including sampling overhead) is large due to the difference between the source and destination network interfaces(Figure 6.23.a). This gap is smaller for the test case with different processor architectures where NIC is not a bottleneck. Yet the improvement is significant.

Overall, both algorithms provide dramatic changes in throughput for disk-to-disk and memory-to-memory transfers. This gap is larger when history information is used. The OPT_B performs better than the OPT_H due to the fact that it uses more resources. Another reason for this difference is the

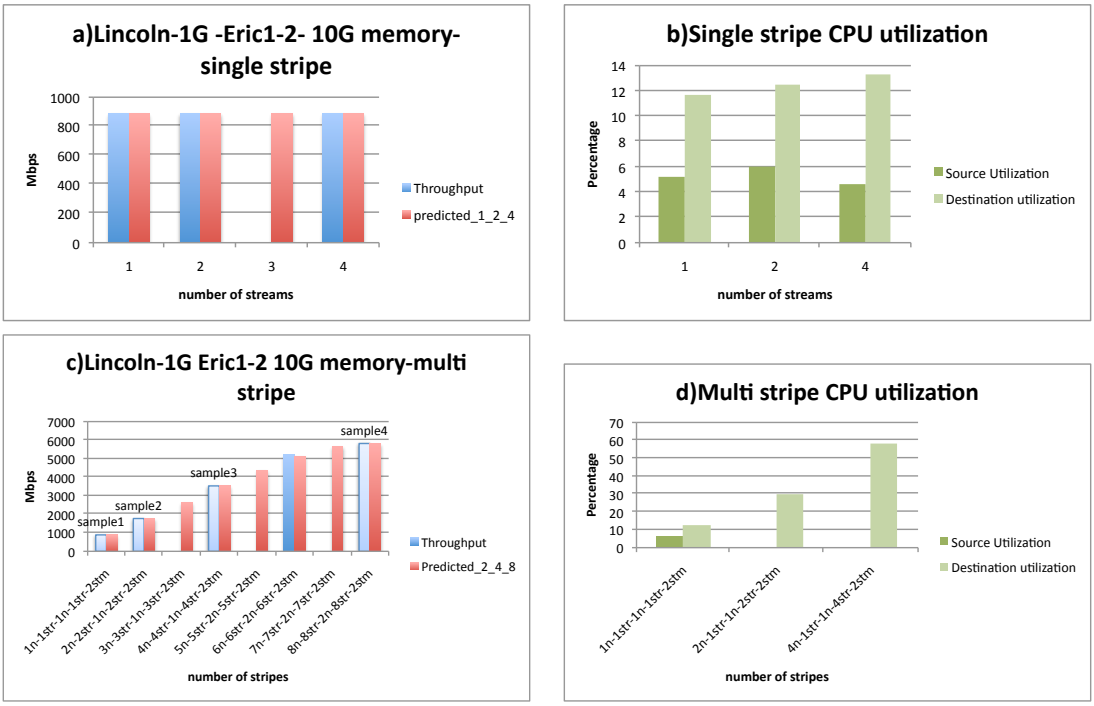


FIGURE 6.21: Model Application (OPT_H) on Memory-to-memory Transfers between Lincoln(1G) and Eric1&2(10G)

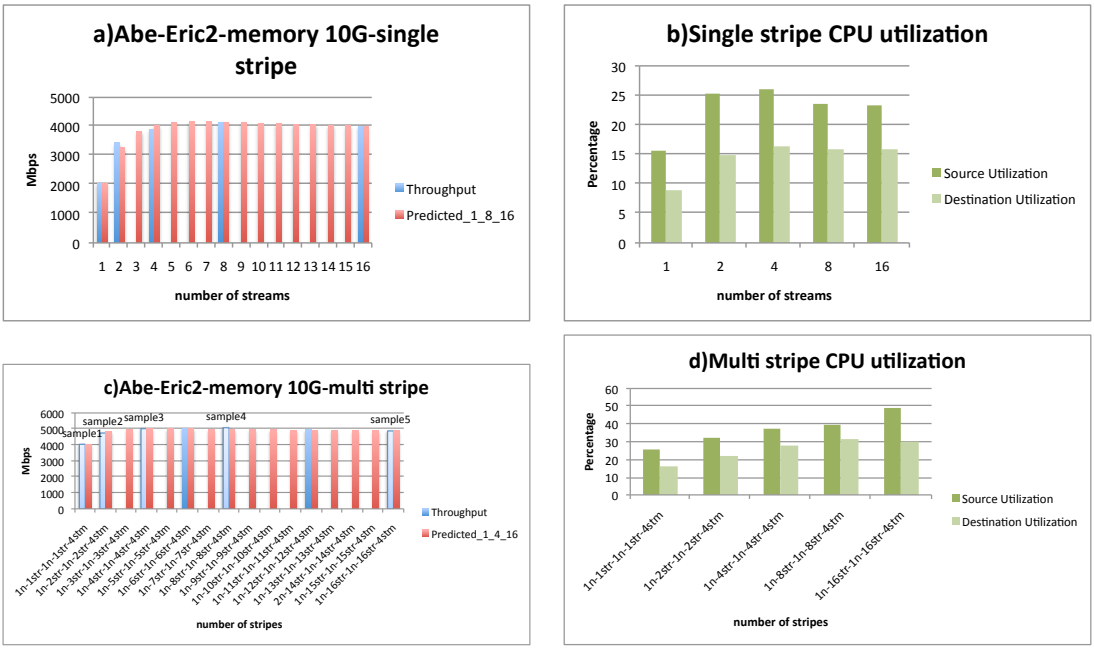


FIGURE 6.22: Model Application (OPT_H) on Memory-to-memory Transfers between Abe(10G) and Eric2(10G)

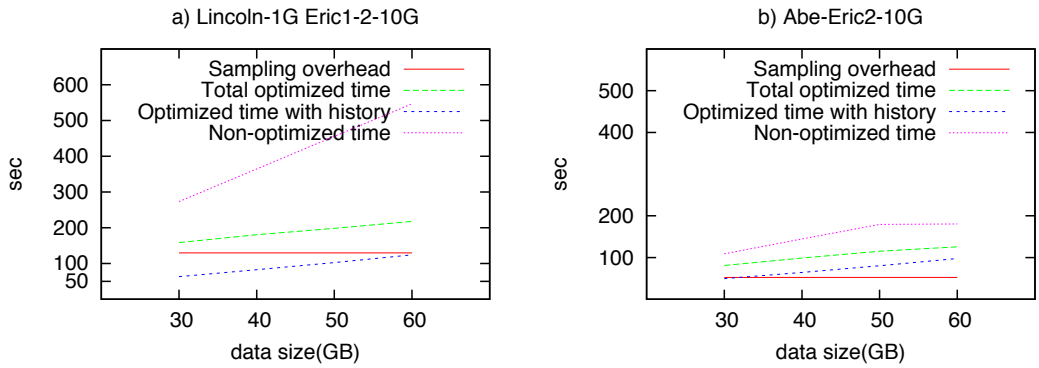


FIGURE 6.23: Transfer times for model application(OPT_H) with 2GB sampling size vs data size in memory-to-memory transfers between NCSA and LONI

unavailability of variety of resources in the OPT_H algorithm where the resource numbers were not sufficient. The prediction applied on the results is very accurate. The throughput achieved ranges between 5-9Gbps including both local and wide area transfers while for disk-to-disk transfers a range of 3-5Gbps is achieved based on the different test cases, load of the network and disk systems.

Chapter 7
Conclusions

In this dissertation, we aimed to address the optimization problem for end-to-end data transfer throughput at the application-level. We analyzed the parallel transfer behavior of a TCP-based data transfer protocol, GridFTP, over different network topologies with different bottleneck link capacities. We presented several prediction models according to the characteristics of the transfers. It has been observed that the aggregate throughput starts to fall down in existence of congestion and none of the existing models could mimic this behavior. By using minimum information on historical results or minimally sampling transfers, our proposed models were able to predict the throughput curve of GridFTP, and we have observed very accurate results. We also conducted a variety of experiments and simulations to see the effect of buffer size tuning over parallel streams. Tuning buffer size and parallel stream number are two ways of optimizing the application level throughput. When the correct balance is found, these techniques applied together could improve the end-to-end throughput comparing to the single application of each technique. Our experiments showed that the usage of parallel streams on fairly tuned buffers could give higher throughput values and less number of streams are needed for optimization.

7.1 Major Contributions

The end-to-end transfer throughput in high-speed networks could be improved dramatically by using data parallelism that takes into account the end-system capacities such as the cpu load, disk access speed and NIC capacity over the nodes. The model and algorithms presented in this dissertation provide the parallelism parameters such as the number of streams per stripe, number of stripe per node and number of nodes for the users. The experimental results conducted using various settings indicate the accuracy of the model and close-to maximal throughput values. The model also gives very good results with immediate sampling especially for large file sizes.

The provided model uses the underlying protocol and does not require any changes in the protocol. The application of the model is easy and it can scale well and is compatible with different architec-

tures. It could be used with any type of node structure and interconnect. Most importantly, it aims maximal achievable throughput by using minimum number of resources.

The experimental results justify our improved model as well as the algorithms to provide improved throughput over high-speed networks and remove the end-system bottlenecks. The time consumed on optimization due to sampling is relatively small compared with the time cost of a file transfer.The difference between the achieved throughput by using this model and the non-optimized throughput is dramatic and the overhead of the model is negligible.

7.2 Future Work

There are several research directions that could be followed after the model presented in this dissertation. The application of the model is practical and does not require much system information. However it makes certain assumptions such as the resources being idle at the beginning of the transfer. A further improved model will take into account the busy resources used by multiple users where a specific allocation is not needed. The accuracy of the model could be improved. A certain sampling size that will give us a good trade-off of accuracy and overhead can help and ease the implementation model. In the case of multiple transfers between different source and destinations submitted to a scheduler, the available optimum throughput predicted by the model could be divided among the possible data transfer jobs.

The algorithms presented in this dissertation work in a fashion that assumes there is no available information about the network capacity and maximum disk capacity. However in some cases, such as dedicated networks, the capacity is known a priori and these algorithms could be simplified based on the same flow model but causing less overhead.

These models could be applied in distributed parallel applications(e.g. distributed visualization, peer-to-peer file sharing, data scheduling) and both adapt to disk-to-disk and memory-to-memory transfers and most importantly scale well no matter how fast the disk systems, the optical network infrastructure or CPU architecture becomes. A future work would be to improve the accuracy and reduce the overhead and provide implementation models that will be appropriate for different application areas.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice Hall, 1993.

[2] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link. The globus striped gridftp server. In *Proc. IEEE Super Computing Conference*, page 54, 2005.

[3] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic. Parallel tcp sockets: Simple model, throughput and validation. In *Proc. IEEE Conference on Computer Communications (INFOCOM'06)*, pages 1–12, April 2006.

[4] H. Balakrishman, V. N. Padmanabhan, S. Seshan, and R. H. Katz M. Stemm. Tcp behavior of a busy internet server: Analysis and improvements. In *Proc. IEEE Conference on Computer Communications (INFOCOM'98)*, pages 252–262, California, USA, March 1998.

[5] K. M. Choi, E. Huh, and H. Choo. Efficient resource management scheme of tcp buffer tuned parallel stream to optimize system performance. In *Proc. Embedded and ubiquitous computing*, Nagasaki, Japan, December 2005.

[6] A. Cohen and R. Cohen. A dynamic approach for efficient tcp buffer allocation. *IEEE Transactions on Computers*, 51(3):303–312, March 2002.

[7] T. Dunigan, M. Mathis, and B. Tierney. A tcp tuning daemon. In *Proc. IEEE Super Computing Conference (SC'02)*, Baltimore, Maryland, USA, November 2002.

[8] L. Eggert, J. Heideman, and J. Touch. Effects of ensemble tcp. *ACM Computer Communication Review*, 30(1):15–29, January 2000.

[9] W. Gropp, E. Lusk, and R. Thakur. *Using MPI-2:Advanced Features of the Message-Passing Interface*. The MIT Press, 1999.

[10] T. J. Hacker, B. D. Noble, and B. D. Atley. The end-to-end performance effects of parallel tcp sockets on a lossy wide area network. In *Proc. IEEE International Symposium on Parallel and Distributed Processing (IPDPS'02)*, pages 434–443, 2002.

[11] T. J. Hacker, B. D. Noble, and B. D. Atley. Adaptive data block scheduling for parallel streams. In *Proc. IEEE International Symposium on High Performance Distributed Computing (HPDC'05)*, pages 265–275, July 2005.

[12] G. Hasegawa, T. Terai, T. Okamoto, and Murata M. Scalable socket buffer tuning for high-performance web servers. In *International Conference on Network Protocols(ICNP'01)*, page 281, 2001.

[13] The lustre file system. http://wiki.lustre.org.

[14] T. Ito, H. Ohsaki, and M. Imase. On parameter tuning of data transfer protocol gridftp for wide-area networks. *International Journal of Computer Science and Engineering*, 2(4):177–183, September 2008.

[15] M. Jain, R. S. Prasad, and C. Davrolis. The tcp bandwidth-delay product revisited: network buffering, cross traffic, and socket buffer auto-sizing. Technical report, Georgia Institute of Technology, 2003.

[16] R. P. Karrer, J. Park, and J. Kim. Adaptive data block scheduling for parallel streams. Technical report, Deutsche Telekom Laboratories, 2006.

[17] G. Kola and M. K. Vernon. Target bandwidth sharing using endhost measures. *Performance Evaluation*, 64(9-12):948–964, October 2007.

[18] T. Kosar and M. Livny. Stork: Making data placement a first class citizen in the grid. In *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, pages 342–349, 2004.

[19] J. Lee, D. Gunter, B. Tierney, B. Allcock, J. Bester, J. Bresnahan, and S. Tuecke. Applied techniques for high bandwidth data transfers across wide area networks. In *Proc. International Conference on Computing in High Energy and Nuclear Physics (CHEP'01)*, Beijing, China, September 2001.

[20] D. Lu, Y. Qiao, and P. A. Dinda. Characterizing and predicting tcp throughput on the wide area network. In *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 414–424, June 2005.

[21] D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante. Modeling and taming parallel tcp on the wide area network. In *Proc. IEEE International Symposium on Parallel and Distributed Processing (IPDPS'05)*, page 68b, April 2005.

[22] Anna Morajko. *Dynamic Tuning of Parallel/Distributed Applications*. PhD thesis, Universitat Autonoma de Barcelona, 2004.

[23] J. Padhye, V. Firoiu, T. Towsley, and J. Kurose. Modeling tcp throughput: a simple model and its empirical validation. *ACM SIGCOMM'98*, 28(4):303–314, October 1998.

[24] R. S. Prasad, M. Jain, and C. Davrolis. Socket buffer auto-sizing for high-performance data transfers. *Journal of Grid Computing*, 1(4):361–376, August 2004.

[25] J. Semke, J. Madhavi, and M. Mathis. Automatic tcp buffer tuning. *ACM SIGCOMM'98*, 28(4):315–323, October 1998.

[26] H. Sivakumar, S. Bailey, and R. L. Grossman. Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks. In *Proc. IEEE Super Computing Conference (SC'00)*, pages 63–63, Texas, USA, November 2000.

[27] N.T.B. Stone, B. Gill, J. Kochmar, R. Light, P.Nowoczynski, J. R. Scott, J. Sommerfield, and C. Vizino. Dmover: Parallel data migration for mainstream users. Technical report, Pittsburgh Supercomputing Center, 2010.

[28] Michael Thomas. Ultralight planets tutorial, 2008.

[29] Brian L. Tierney. Tcp tuning techniques for high-speed wide-area networks, nfnn2 talk, 2005.

[30] L. Torvalds and The Free Software Company. The linux kernel. http://www.kernel.org.

[31] Advanced networking for distributed petascale science. Technical report, US Department of Energy, apr 2008.

[32] Distributed parallel storage server. http://www-didc.lbl.gov/DPSS/.

[33] Wireshark. http://www.wireshark.org.

[34] E. Weigle and W. Feng. Dynamic right-sizing: A simulation study. In *Proc. IEEE International Conference on Computer Communications and Networks (ICCCN'01)*, 2001.

[35] E. Weigle and W. Feng. A comparison of tcp automatic tuning techniques for distributed computing. In *Proc. IEEE International Symposium on High Performance Distributed Computing (HPDC'02)*, 2002.

[36] Louisiana optical network initiative. http://www.loni.org.

[37] The teragrid. http://www.teragrid.org.

Vita

Esma Yildirim was born in Istanbul, Turkey, in 1982. She graduated from Fatih University, Istanbul in 2004 with a bachelor's degree in computer engineering with Honors and 1st rank. After graduation she started the master's program in Marmara University Computer Engineering Department in 2004. She worked in Avrupa Software Company until 2005 and was part of a ERP software development group. In 2005 she started teaching in Fatih University Vocational School in the Department of Computer Technologies and Programming. She taught courses such as database systems, algorithms, introduction to electronics and programming. She received her master's degree in computer engineering in 2006 with a focus on "Scheduling in Distributed Heterogeneous Environments".

She was offered a graduate research assistantship in Louisiana State University, Center for Computation and Technology in 2006 and started the doctoral program in the Computer Science Department the same year. She has worked on several research projects (Petashare, Stork, Cybertools), has won several student scholarships (OGF, Ultralight/PlanetS), attended and presented in several conferences.

Her research interests are grid computing, data challenges in high-speed networks, scheduling in distributed environments and high-performance computing. She has published 8 peer-reviewed articles, has written 2 book chapters and presented several posters in different conferences and meetings.

Her articles and publications include:

- Yin, D., E. Yildirim and T. Kosar. 2011. A Data Throughput Prediction and Optimization Service for Widely Distributed Many-Task Computing. To appear in IEEE Transactions on Parallel and Distributed Systems- Special Issue on Many Task Computing

- Yildirim, E. , D. Yin and T. Kosar. 2010. Prediction of Optimal Parallelism Level in Wide Area Data Transfers . To appear in IEEE Transactions on Parallel and Distributed Systems.

- Kosar, T., M. Balman, I. Suslu,E. Yildirim, and D. Yin. 2009. Data-Aware Distributed Computing with Stork Data Scheduler. Proceedings of SEE-GRID'SCI'09, Istanbul, Turkey.

- Yin, D., E. Yildirim and T. Kosar. 2009. A Data Throughput Prediction and Optimization Service for Widely Distributed Many-Task Computing. Proceedings of Many Task Computing in Grids and Supercomputers (MTAGS 2009) in conjunction with SC09, Portland, Oregon.

- Yildirim, E. , D. Yin and T. Kosar. 2009. Balancing TCP Buffer vs Parallel Streams in Application Level Throughput Optimization. Proceedings of International Workshop on Data-Aware Distributed Computing (DADC 2009) in conjunction with HPDC09, Munich, Germany.

- Yildirim, E., I.H. Suslu and T. Kosar. 2008. Which Network Measurement Tool is Right for You? A Multidimensional Comparison Study. Proceedings of IEEE/ACM Int. Conference on Grid Computing (Grid 2008), Tsukuba, Japan

- Yildirim, E., M. Balman and T. Kosar. 2008. Dynamically Tuning Level of Parallelism in Wide Area Data Transfers. Proceedings of International Workshop on Data-Aware Distributed Computing (DADC 2008) in conjunction with HPDC08, Boston, MA

- Yildirim, E., H. Topcuoglu and T. Kosar. 2007. A Memetic Algorithm for Reliability-based Dynamic Scheduling in Heterogeneous Computing Environments. Proceedings of 19th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2007), Cambridge, MA

Her book chapters include :

- Yildirim, E., T. Kosar. End-to-end Data-Flow Parallelism for Transfer Throughput Optimization. Accepted as a book chapter in Advancements in Distributed Computing and Internet Technologies

- Yildirim, E., T. Kosar. Application-level Tuning of End-to-end Data Transfer Throughput. Accepted as a book chapter in Data Intensive Distributed Computing: Challenges and Solutions for Large-scale Information Management

Her conference and poster presentations include:

- Yildirim, E., I.H. Suslu and T. Kosar. 2008. Which Network Measurement Tool is Right for You? A Multidimensional Comparison Study. Proceedings of IEEE/ACM Int. Conference on Grid Computing (Grid 2008), Tsukuba, Japan

- Yildirim, E., H. Topcuoglu and T. Kosar. 2007. A Memetic Algorithm for Reliability-based Dynamic Scheduling in Heterogeneous Computing Environments. Proceedings of 19th IASTED Inter- national Conference on Parallel and Distributed Computing and Systems (PDCS 2007), Cambridge, MA

- Yildirim., E, D. Yin and T. Kosar. 2008. Predicting Optimal Parallelism Level in Wide Area Data Transfers. Louisiana Cyberinfrastructure and Science Drivers Symposium, Baton Rouge, LA (Best Student Poster Award)

- Yildirim., E, D. Yin and T. Kosar. 2009. A Parallel TCP Throughput Optimization Service. Open Science Grid All Hands Meeting, LIGO, Baton Rouge, LA (Poster)

At the December 2010 Commencement, she will receive the degree of Doctor of Philosophy. After graduation she will work in the United States as a research scientist.