University of Nevada, Reno

Real Time Throughput Estimation and Optimization

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in
Computer Science and Engineering

by

Bahadir Ali Pehlivan

Engin Arslan/Thesis Advisor

May, 2019

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

BAHADIR ALI PEHLIVAN

entitled

Real Time Throughput Estimation and Optimization

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Engin Arslan, Advisor

Emily Hand, Committee Member

Ilya Zaliapin, Graduate School Representative

David W. Zeh, Ph.D., Dean, Graduate School

May 2019

ABSTRACT

Obtaining optimal data transfer performance is of utmost importance to today's data-intensive distributed applications and wide-area data replication services to meet stringent networking demands. Tuning application-layer protocol parameters such as pipelining, parallelism, and concurrency can significantly increase efficient utilization of the available network bandwidth as well as the end-to-end data transfer performance. However, determining the best settings for these parameters is a challenging problem, as network conditions can vary greatly between sites and over time. Poor protocol tuning can cause either under- or over-utilization of network resources and thus degrade transfer performance. Real-time throughput estimation and transfer optimization approach offer promising solutions as it can discover optimal transfer configuration in the run time without requiring an upfront work or making assumptions about underlying system architectures.

In this thesis, we use a real time approach for efficient transfer optimization by offering a heuristic solution for quick search space exploration and reduced overhead of evaluating various configuration settings. In the first work, we developed a real time tuning method for optimizing the number of concurrent transfers to yield high throughput while keeping the system overhead at minimal. Real time tuning runs a series of sample transfers with different concurrency values to identify the value after which throughput increase becomes negligible. When it is compared with other non real time methods it is much more flexible to adapt different conditions. It was compared with a fixed size sampling method whose sample transfer size is decided by a model trained by previous transfer data. That method on average requires 20% of the data to be sampled and this is too much to afford for transferring large data. It was also compared with fixed time sampling method and it is seen that the fixed time method cannot adapt unexpected changes in network condition. To further improve the benefit of real time tuning solutions, in our second work we examined sample

transfers to minimize searching time while not degrading accuracy. We evaluated several time-series and regression models to be able to quickly determine throughput convergence time. The results gathered in various networks with rich set of transfer configurations indicate that, in most cases, Autoregressive model can accurately estimate sample transfer throughput in less than 5 seconds. This is up-to 4x improvement over the state-of-the-art solution. We also realized that while the most common transfer applications report instantaneous throughput in at most every second, decreasing report intervals to orders of 100s of milliseconds is key to further reduce execution time.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Data is becoming larger and more prevalent with new large scale projects all around the world and wide usage of big data increases the need for high speed networks. Big scientific experiments such as environmental and coastal hazard prediction [1], climate modeling [2], and high-energy physics simulations [3] generate data volumes reaching petabytes per year. This huge volume of data is often moved to remote sites for various purposes such as processing, collaboration, and archival. An example scenario might be the need to backup the data which is captured by an imaging satellite in remote locations. For instance, Event Horizon Telescope [4] runs data collection using telescopes located in Hawaii and Mexico, mountains in Arizona and the Spanish Sierra Nevada, the Chilean Atacama Desert, and Antarctica. High-quality images taken by these satellites are then shipped to a high-performance computing cluster to run complete data analysis.

Even though high speed networks with up-to 100 Gbps network bandwidth have been established, many users still experience difficulty reaching the theoretical maximum throughput, causing under-utilization of resources [5]. Some common reasons for this are underutilized end-system processor performance, poor file system performance, ill-designed transfer protocols, bad configuration of end hosts, low disk I/O speeds, server implementations not taking advantage of parallel I/O opportunities, background traffic at inter-system routing nodes, and unsuitable system-level tuning of networking protocols. The effects of some of these factors can be mitigated to varying degrees through the use of many available techniques. However, too little use of one technique might cause resource underutilization whereas excessive usage might overburden end hosts and network. Furthermore, the optimal level of usage for each technique varies depending on the network and end-system

conditions, meaning no single parameter combination is optimal for all different scenarios.

In this thesis, we focused on real time throughput optimization and estimation for data transfer. In the first work, we developed a method for real time tuning of concurrency parameter [6]. By tuning the concurrency parameter, our method seeks for the optimal point which leads to best ratio of throughput over resource usage. So, while reaching the best throughput is the main motivation, it is also important to not impose much burden on the end hosts and network. This work basically tries different concurrency values in an order which is designed to take less time to find optimal concurrency. To assess utility of these different concurrency values, some sample transfers are done for each concurrency value. Even though sample transfers cause under-utilization of network until the optimal concurrency is found, overall the method reaches better average throughput. After sample transfers are done and different concurrency parameters are assessed, the algorithm continues the rest of the transfer with a concurrency value which is considered as optimal.

If assessment of different concurrency values can be done faster, then our real time tuning method would converge quicker. Also this fast assessment method can enhance any method which requires estimation of average throughput in real time. Hence, in the second work, we evaluated several time series and regression models all of which processes sample transfer throughput values in real-time and predicts expected transfer throughput within a few seconds [7]. Beside doing it quickly, the estimation precision is also important to make accurate decisions. The first work is trying to fix concurrency parameter whereas the goal of the second work is estimation of expected average throughput. In the first work, even though we acquired enhancement over previous works, the time required to assess throughput value for a concurrency parameter was not quick enough for a real time application. Hence, second work aims to minimize the cost of running sample transfers. Both works target real time solutions without using any historical network transfer data or equations with system related parameters.

CHAPTER 2
RELATED WORK AND BACKGROUND

2.1 Throughput Optimization

Most of the existing work on throughput optimization has been at the transport layer, including designing new transport protocols [8, 9, 10, 11, 12, 13]. Other approaches aim to improve throughput by opening flows over multiple paths between end-systems [14, 15]. However, these solutions generally face deployment challenges due to hardware/software requirements. Besides, improving TCP's congestion control will not solve all the aforementioned problems since high-speed end-to-end transfers also require fast data transmission from/to end servers and storage systems. At a higher level, techniques have been developed that focus on keeping the existing underlying protocol intact and tuning it at the application level for improved performance.

One common way to address low-performance problems is through tuning at the application level by tuning parameters such as pipelining [16], parallelism [17], concurrency [6], buffer size [18], block size [19], and striping [20]. These parameters can be tuned at the application layer without the need to change the underlying transfer protocols and can significantly improve the end-to-end data transfer performance. Among these, pipelining, parallelism, and concurrency are found to be the most effective parameters to address the majority of underlying performance issues [21, 22, 5, 23, 24, 25]. Furthermore, out of these three, concurrency is more effective and this was our basis for the first work in which we implemented real time tuning of concurrency [6, 26].

Below we give definition and some insight for these three parameters:

- **Pipelining**: Pipelining targets the problem of transferring a large numbers of small

files [27, 28, 29]. In most control channel-based transfer protocols, transfer of a file must complete and be acknowledged before the next file can be requested. This may cause a delay of more than one RTT between individual file transfers especially in cases where the client is located far from source and destination (i.e., third-party transfers). With pipelining, multiple transfer commands can be queued up at the server, reducing the delay between transfer completion and receipt of the next command from the client. It is especially helpful for transferring lots of small files as it can eliminate the idle time between consecutive file transfers.

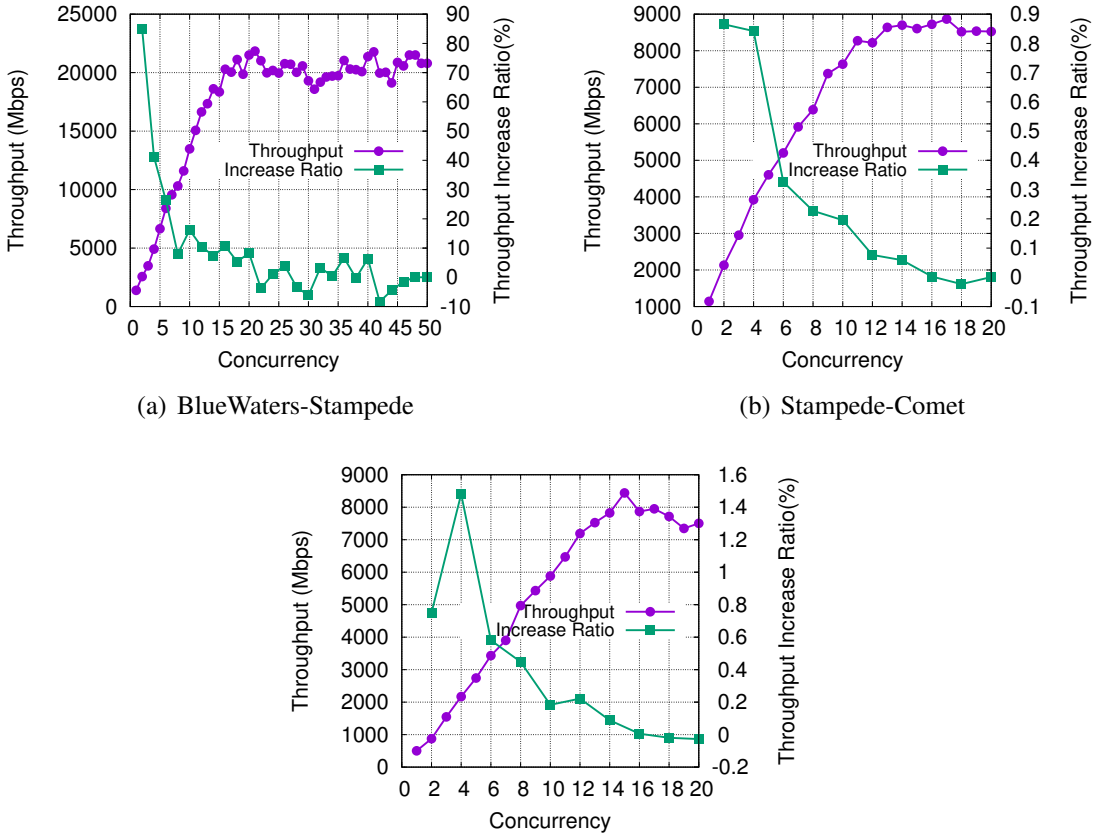
- **Parallelism**: Parallelism is the parameter which decides on how many streams will one file be divided over a network. For example, when parallelism value is five, that means a file is sent through five data channels. This method is especially useful for larger files when buffer size is inadequate for fully utilizing available bandwidth. However, it can incur overhead for small files as dividing small files over multiple channels leads to transferring negligible amount of data in each channel, causing underutilization of channels. Even for large files, using more than enough parallel streams would lead increased overhead and degraded transfer performance. Hence, it should be carefully set with respect to file types in a dataset. It also helps to use available bandwidth more efficient in the beginning of the transfer. A single stream will use a small part of bandwidth due to slow start phase of TCP but several streams will increase the usage of bandwidth in this phase.

- **Concurrency**: Concurrency is the number of files which are transferred at the same time. While parallelism created multiple network channels for single file, concurrency uses channels to transfer different files. Also, concurrency creates multiple processes for reading files from storage while parallelism uses only one process for disk I/O. This can have major impact on the performance when end systems have parallel file systems by increasing disk I/O performance as a result of utilizing accessing

to storage servers to read/write files.

While significant performance gains can be achieved by tuning application-level protocol parameters, the optimal value of these transfer parameters depends on many factors including dataset (i.e., file size and the number of files), network (i.e., bandwidth, round-trip-time, and background traffic on network), and end-system characteristics (i.e., file system and transfer protocol). Thus, finding the best parameter combination is a challenging task due to large search space and prohibitive cost of exhaustive profiling. Figure 2.1 shows the impact of the one of the application-layer transfer parameters, concurrency, on throughput for different transfer conditions. It is clear that different transfers reach peak throughput at different concurrency values. This shows transfer specific tuning of concurrency might be the best option instead of coming up with a universal formula. In the same graph, although it seems like throughput stays relatively stable after reaching to its peak, using large values for concurrency will cause other issues beyond transfer throughput such as increased load at the end servers and network, higher energy consumption and unfair resource allocation as a result of opening too many network connections. Hence, it is essential to explore the optimal settings that achieve high transfer throughput while not imposing too much overhead.

Several solutions were proposed to improve utilization of a single path by means of parallel streams [30, 31, 32, 33], pipelining [27, 28, 34],and concurrent transfers [35, 36]. Several file transfer tools tried to statically tune a subset of these parameters in an effort to improve the end-to-end data transfer throughput [37, 38, 39]. Liu et al. [36] developed a tool to determine the optimal number of concurrent file transfers based on observed transfer throughput. While the presented performance increase is significant, setting the number of concurrent transfers just by considering transfer throughput would lead to opening too many processes and connections hence overloads the end system and network. Thus, both transfer throughput and system overhead needs to be considered when tuning concurrency.



(a) BlueWaters-Stampede

(b) Stampede-Comet

(c) SuperMIC-Bridges

Figure 2.1: Effect of concurrency value on transfer throughput

We propose a real time tuning algorithm for concurrency using a cost function which rewards high throughput and low concurrency values.

In addition, several approaches were proposed to tune multiple transfer parameters at the same time using heuristics [21, 5], offline modeling [40, 24, 23], and adaptive [41] techniques. Globus Online [21] sets pipelining, parallelism, and concurrency parameters to pre-determined values for three different file sizes (i.e., less than 50MB, larger than 250MB, and in between). However, the protocol tuning Globus Online performs is non-adaptive; it does not change depending on dataset and network settings and does not perform well in various scenarios.

Modeling based approaches use historical data to derive models to define a relationship

between transfer parameters and throughput. HARP [24, 23] derives polynomial models to relate application-layer transfer parameter to transfer throughput and solves these models for maximum throughput to find the corresponding values for transfer parameters. While it can obtain close-to-optimal throughput results, it requires historical data to contain up-to-date information for different network conditions (such as background traffic) which hinders its deployment.

In addition to throughput optimization, researchers also studied integrity verification in the context of storage outsourcing [42, 43, 44], long term archiving [45, 46], file systems [47, 48, 49], databases [50], provenance [51], and data transfer [52]. Zhang et al. [49] evaluated Zetabyte Files System (ZFS) in terms of robustness to disk and memory fault injections. It has been found that while ZFS is able to detect and mostly recover from disk corruptions, it is susceptible to memory corruptions since it does not check the integrity of data blocks when they reside in the memory.

Globus [53] supports end-to-end integrity verification for data transfers. It pipelines data transfers and checksum computation to minimize the overhead of integrity verification. However, its pipelining approach fails to work well when a dataset consist of mixed file sizes. Liu et al. propose block-level pipelining to improve pipelining of mixed size datasets by dividing large files into blocks [52]. It reduces execution time considerably especially when dataset is composed of files with mixed sizes, however it requires careful tuning of block size to perform well. Arslan et al. we proposed Fast Integrity Verification Algorithm (FIVER) that reads files once and run the transfer and checksum computation processes simultaneously, reducing I/O overhead and checksum computation time [54]. FIVER outperformed state-of-the-art solutions by reducing the overhead of integrity verification from up-to 60% to less than 10%. Charryev et al. proposed robs integrity verification algorithm that can capture silent disk write errors by clearing memory cache periodically [55].

Impact of Application Level Parameters on Throughput

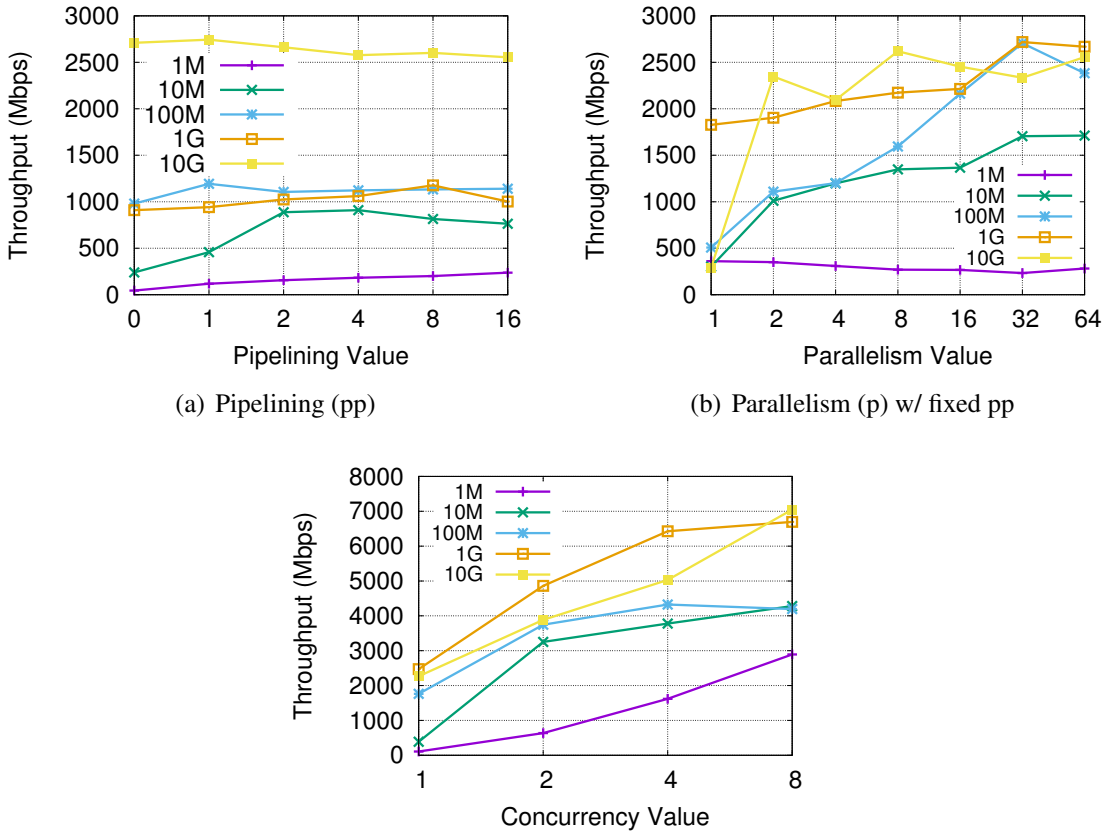
Specs	Storage	CPU	Memory (GB)	Bandwidth (Gbps)	RTT (ms)
Lonestar-Gordon	Lustre	10 x Intel @ 2.3 GHz 2 x Intel @ 2.6 GHz	128 & 64	10	60
BlueWaters-Stampede	Lustre	2 x AMD @ 2.3 GHz 4 x Intel @ 1.4GHz	64 & 32	3x10	32
Supermic-Bridges	Lustre	2 x Intel @ 2.8 GHz 2 x Intel @ 3.3 GHz	128 & 128	10	45
Stampede-Comet	Lustre	4 x Intel @ 1.4 GHz 24 x Intel @ 3.2 GHz	32 & 64	10	40

Table 2.1: System specification of testbeds.

Pipelining, parallelism, and concurrency play a significant role in affecting achievable transfer throughput. However, setting the optimal levels for these parameters is a challenging problem, and poorly-tuned parameters can either cause underutilization of the network or overburden the network and degrade the performance due to increased packet loss, end-system overhead, and other factors. These transfer parameters can be set to any integer value, however system administrators may define upper limits as too large values may cause system to crash.

To analyze the effects of pipelining, parallelism and concurrency on the transfer of different file sizes, for each of the parameters some experiments were conducted separately, as shown in Figure 2.2. These experiments were run on XSEDENet [56], production-level high-bandwidth network, using two supercomputers whose specifications are given in Table 2.1.

Five datasets were generated for five different file sizes and transferred each dataset only by changing one parameter (i.e., pipelining, parallelism or concurrency) at a time to observe the individual effect of each parameter. Figure 2.2(a) shows that pipelining can increase the throughput of small files by up to 2x while its impact becomes negligible for large files. On the contrary, parallelism helps to improve transfer throughput for large files significantly by overcoming buffer size limitation while it has no impact (if not negative) on small files as shown in Figure 2.2(b). Thus, it is necessary to enable different parameters



(a) Pipelining (pp)

(b) Parallelism (p) w/ fixed pp

(c) Concurrency (cc) w/ fixed pp and p

Figure 2.2: Effect of protocol parameters on transfer throughput for different file sizes in XSEDE (Lonestar-Gordon).

for different file sizes which requires separating small and large files if a given dataset contains both small and large files. Moreover, some of the datasets exhibit similar behavior to changing parameter values. For example, pipelining has either limited or negative impact on 1 GB and 10G datasets as shown in Figure **??**. Similarly, the throughput for both datasets increases with increased parallelism and concurrency levels.

Concurrency is the most broadly effective parameter for all file sizes in both networks as it helps to improve disk I/O throughput by means of reading/writing multiple files simultaneously. While it is the most effective parameter, it incurs the most overhead to the end systems at the same time by increasing CPU usage at the end systems [57, 58]. So, the value for concurrency should be decided carefully.

2.2 Sample Transfer Modeling

As mentioned before, there have been several approaches to tune some of the application-layer parameters in order to maximize transfer throughput using heuristics [21], supervised [22, 40], semi-supervised [59, 23] and unsupervised [19, 60, 61, 25] models. Since heuristic, supervised, and semi-supervised models require a significant upfront work and re-adjustments when system configuration changes, unsupervised methods such as online optimization offer promising approach as they can adapt to changing network conditions by discovering the optimal transfer settings in the real-time. They do this by running a series of sample transfers to evaluate different parameter configurations and swiftly converge to the optimal setting. Thus, their success heavily rely on the accuracy and cost of sample transfers. Yet, shared nature of high performance networks and end system resources leads to significant fluctuations in transfer throughput, hindering fast and accurate estimation of average sample transfer throughput. In addition to its importance for real-time transfer parameter tuning algorithms, sample transfers are also used in network monitoring [62], workflow scheduling [63], and adaptive video streaming [64].

Specs	**Storage**	**CPU**	**Memory (GB)**	**Bandwidth (Gbps)**	**RTT (ms)**	**File Count**
XSEDE (Stampede2-Comet)	Lustre	28 x Intel @ 2.6 GHz 24 x Intel @ 3.2 GHz	96 64	10	40	28,209
ESnet	RAID-0	12 x Intel @3.4 GHZ	128	100	89	5,218
Pronghorn	GPFS	16 x Intel @2.1 GHz	192	10	0.1	2,316
HPCLab	NVMe SSD	16 x Intel @2.6 GHz	64	40	0.1	16,383

Table 2.2: System specification of networks used in experiments of second work.

Current solutions to run sample transfers include fixed data size [65], fixed-time duration [66], and adaptive approach [23]. In the fixed data size approach, a pre-calculated amount of a dataset (e.g., 10%) is used to run sample transfers, however it requires an upfront work to determine the optimal data size which may not be feasible for every transfer operation [65]. On the other hand, the fixed-time approach requires a fine tuning of time

duration that sample transfers will run, otherwise it may also result in poor accuracy or take too long to finish. Among them, adaptive approach promises fast convergence with highest accuracy, however we found that it can fail to converge when transfer throughput exhibit moderate to high fluctuations.

Previous works found that throughput of data transfers in shared networks rely on various factors including network and end system interference, dataset characteristics, network settings, and transfer configurations [59, 6, 23, 5, 22]. So, it is nearly impossible to predict transfer throughput of a dataset in advance. Thus, sample transfers are used to sense expected throughput which can be used to tune transfer configurations to maximize transfer throughput or to adjust execution plan for distributed workflows. The most common way to run sample transfers is to use a portion of original dataset such that sample transfers could also contribute to ultimate transfer goal. However, there is no consensus on how to schedule sample transfers as obtaining accurate and timely results is a diffucult task due to unpredictable nature of resource interference.

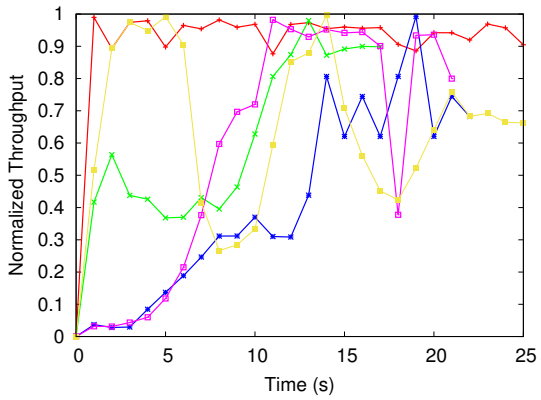


Figure 2.3: Convergence behavior of file transfers throughput follows distinct pattern based on network settings, dataset characteristics and background traffic

Figure 2.3 demonstrates how normalized instantaneous throughput fluctuates over time for different transfers in different networks. It is clear in the figure that instantaneous throughput values exhibit sharp fluctuations over time in addition to convergence times be-

ing different. While one transfer can reach to maximum throughput in as short as 3 seconds, it can take up to 20 seconds for another one. While running sample transfer for the worst-case scenario might sound like a possible solution, it would significantly deteriorate transfer optimization algorithms that aims to find optimal transfer configuration by running sample transfers in the runtime. Thus, it is nontrivial to schedule sample transfers that can capture fast and accurate results without imposing so much overhead. Researchers proposed fixed-size [65], fixed-duration [66], adaptive [59], and modelling-based [65] techniques to conduct sample transfers.

Yildirim et al. developed a model to estimate percentage of dataset size to be used in sample transfers [65]. They first run extensive experiments to collect accuracy statistics for various sampling sizes and then run regression analysis to extract the relationship between sampling size and network and dataset settings such as bandwidth, RTT, and average file size. The model is then used to predict optimal sampling size for future transfers. As the model's accuracy heavily depends on the collected data logs, it requires significant upfront work to perform well. Moreover, the model estimates sampling size to be between 10% and 23% of dataset size which would incur too much delay for large datasets and result in inaccurate sampling size for small datasets. For example, suppose we have a dataset which takes 1000 seconds to be transferred completely, then 23% of the dataset will take around 230 seconds to be transferred which is unnecessarily too long.

In another work, throughput of sample transfers are determined by running them for a fixed time duration [66]. The goal is to evaluate different transfer configurations and determine the one with the highest throughput to energy consumption ratio. They concluded that 5 seconds is sufficient to predict the performance of any transfer configuration in their test networks. However, running sample transfers for fixed time period is sensitive to network conditions as it would be too short for networks with high speed and RTT, and too long for low-bandwidth local area network experiments. Indeed, we have observed in our

experiments that transfer convergence speed could take up to 20 seconds in some networks due to slow connection setup and high bandwidth-delay product.

Another method is adaptive sample transfer which starts by transferring an entire dataset and monitor instantaneous throughput periodically. If throughput of two consecutive monitor intervals are closer than a defined threshold, it stops the transfer and take the average throughput of last two intervals as the throughput of the sample transfer. Adaptive approach works well if the throughput data is in an ideal pattern with less fluctuations. But in normal network scenario there are fluctuations in both transfer throughput and there might be more than one closeness value, one in the beginning and one few seconds after that.

In another study [67], several direct search methods are applied to tune parameters during transfer. They use control epochs to do changes on parameters which affect the throughput. So any change should be done at the end of a control epoch. In the experiments they have done, control epoch is set to 30 seconds which is a bit higher when compared to our sampling time.

Karrer et al. use regression and autoregressive prediction models to estimate throughput of a TCP connection [68]. Autoregressive models yield a relatively good accuracy in short time but search space for best parameters is large in their work and hard to use in adaptive applications. Also, they only do prediction for single TCP flow whereas we do it for several TCP flows in our work.

CHAPTER 3
REAL-TIME OPTIMIZATION OF CONCURRENCY VALUE

In this work, we developed a method which does real time tuning of concurrency value [6]. Real time tuning works by iterating different concurrency values in the beginning of transfer and choosing the best out of them to continue to transfer. It is compared with heuristic algorithms which decide concurrency, parallelism and pipelining values beforehand by using some equations. Heuristic approach uses these equations and calculates values for these parameters and keeps them the same throughout the transfer whereas real time tuning changes concurrency value until the algorithm decides on a value as optimal. Beside heuristic approach, this real time tuning method is also compared to a previous work [24] which does tuning based on historical data and real time tuning showed promising results. Even though it takes some time to tune the concurrency value in run time still it succeeded to perform better than previous methods overall. Also, in our second work, methods for decreasing this tuning time is discussed which may make this method even better.

3.1 Clustering Dataset Based on File Size

Heuristic equations are used to decide on parallelism and pipelining values before the transfer starts. For concurrency value real time tuning is used which will be detailed later. Those heuristic equations use average file size, Bandwidth delay product, TCP buffer size and $maxCC$ value (which is given by administrator). So it takes several parameters into account but decision is made once in the beginning and values calculated are not changed during the transfer.

Also three clustering algorithms were used in this paper: (i) the “Single-Cluster (SC)” algorithm, which separates files into clusters based on file size, and then transfers each clus-

ter with its optimal parameters; (ii) the "Multi-Cluster (MC)" algorithm, which likewise creates clusters based on the file size, but rather than scheduling each cluster separately, it transfers multiple file clusters together in order to minimize the effect of poor performance of small file transfers; (iii) the "Pro-Active Multi-Cluster (ProMC)" algorithm, which transfers multiple clusters at the same time similar to MC, but instead of allocating data transfer channels equally among clusters, considers chunk size and type, and improves the performance especially if the small files dominate the dataset. Although the proposed algorithms differ in terms of how to schedule clusters, they share methods that determine the number of clusters and parameter values for each cluster. While doing comparisons we used our real time tuning method with MC algorithm and other compared methods are used with MC too.

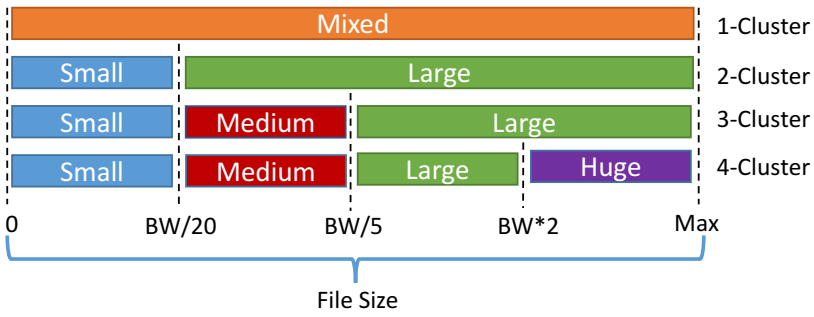


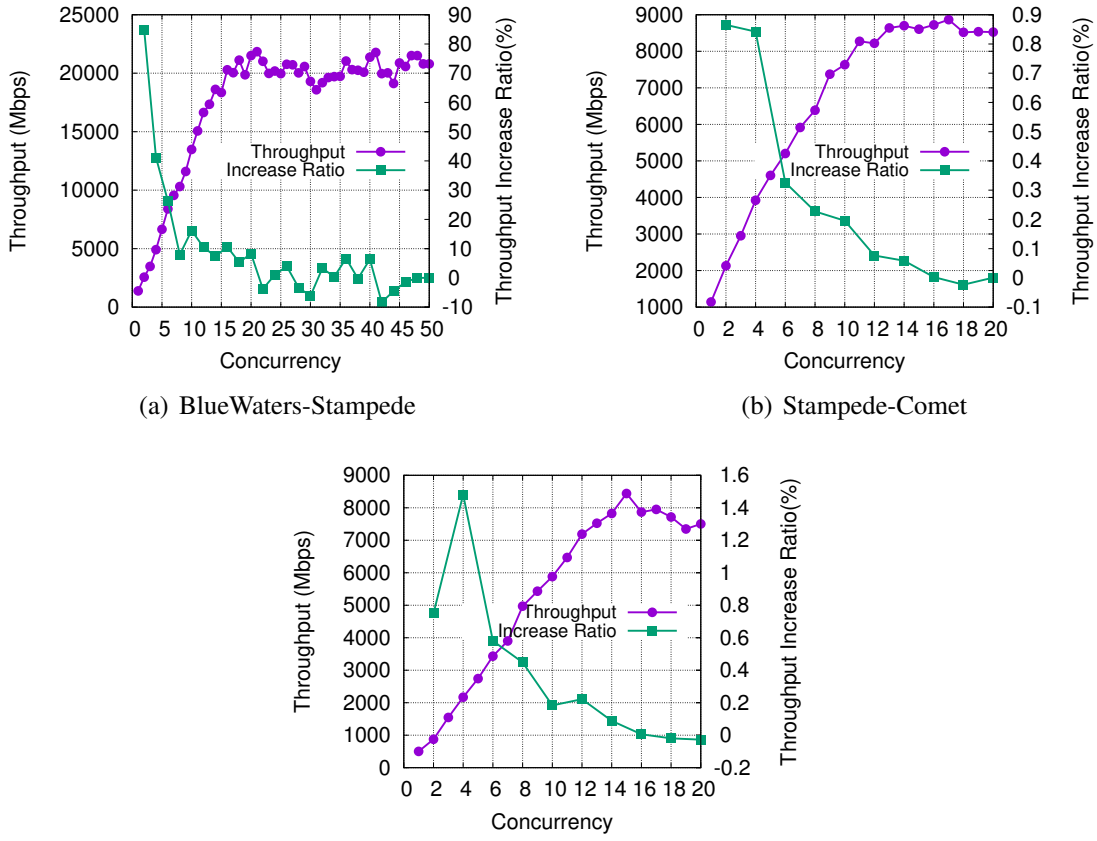
Figure 3.1: Cut-off points to partition datasets into clusters (BW = network bandwidth).

Algorithm 1 — Multi-Cluster (MC) Scheduling

```
1: function TRANSFER(source, destination, BW, RTT, maxCC)
2:     BDP = BW * RTT
3:     allFiles = fetchFileListFromSource()
4:     clusters = partitionFiles(allFiles, BDP)
5:     foreach cluster in clusters do
6:         cluster.optParams = findOptimalParameters(chunk)
7:     end foreach
8:     while maxCC > 0 do
9:         cluster = getNextChunk()          ▷ Round-robin from set of {Huge,Small,Large,Middle}
10:        cluster.concurrency += 1          ▷ Add channel to the chunk
11:        maxCC -= 1
12:    end while
13:    foreach cluster in clusters do
14:        transfer(cluster, cluster.concurrency)   ▷ Run in the background
15:    end foreach
16:    reAllocateFinishedChannels()
17: end function
```

3.2 Real Time Optimization Method

The experiments done clearly showed that concurrency is the most effective parameter in transfer throughput. Yet, it incurs the most overhead to the end systems and network, thus needs to be adjusted carefully. Since the optimal value of concurrency is not same in different networks, the value of $maxCC$ (maximum allowed concurrency) needs be tuned for each network which requires domain expertise. Hence, we proposed real time optimization algorithm for concurrency to find its optimal value which would attain highest throughput for the transfer while not causing too much overhead.



(a) BlueWaters-Stampede

(b) Stampede-Comet

(c) SuperMIC-Bridges

Figure 3.2: While performance of MC increases initially, it stabilizes at some point for increasing concurrency (maxCC) values

Algorithm 2 — Calculation of protocol parameter values by heuristic equations

1: **function** FINDOPTIMALPARAMETERS(avgFileSize, BDP, bufferSize, maxCC)
2: $pipelining = \frac{BDP}{avgFileSize}$
3: $parallelism = Min(\lceil \frac{BDP}{bufferSize} \rceil, \lceil \frac{avgFileSize}{bufferSize} \rceil)$
4: $concurrency = Min(Max(\frac{BDP}{avgFileSize}, 2), maxCC)$
5: **return** (pipelining, parallelism, concurrency)
6: **end function**

Theoretically, concurrency may take any value greater than one, however setting it to very large values would cause too much overhead to end systems and network. So, we limited its maximum value to 32. However, finding optimal value between 1 to 32 in a reasonable amount of time is still a challenging task. Using brute-force to find the optimal value would be too time consuming and only work for very large transfers that lasts over hours. Thus, we first measured the performance of different concurrency values in different networks using MC algorithm as shown in Figure 3.2. The value of pipelining and parallelism are still determined by the heuristic given in Algorithm 2. Increase ratio is defined by percentage of increase between two consecutive concurrency values (when step size is 2). The results of all three networks show that while throughput increase ratio is high initially, it falls sharply with growing concurrency values. Hence, it is important to identify sweet spot for concurrency after which throughput increase becomes negligible. While sweet spot is different in all three networks, it mostly happens when increase ratio falls below 10%. So, we implemented a search algorithm, to find optimal concurrency value using increase ratio to compare the performance of different concurrency values.

$$\gamma(cc_1, cc_2) = \frac{\frac{thr(cc_2) - thr(cc_1)}{thr(cc_1)}}{\frac{cc_2}{cc_1} * tolerance} \tag{3.1}$$

The real time optimization algorithm consists of three phases; FAST_INCREASE, SEEKING and FINISHED phases as shown in Algorithm 3. It starts with FAST_INCREASE phase and runs until it reaches to FINISHED phase. In FAST_INCREASE and SEEK-

Algorithm 3 — Searching for optimal concurrency value in run-time

```
1: function FINDOPTIMALCONCURRENCY(currentCC)
2:     while phase is not FINISHED do
3:         if phase == FAST_INCREASE then
4:             if compareUtility(currentCC, currentCC/2) > 1 and currentCC * 2 < UPPERLIMIT then
5:                 currentCC = currentCC * 2
6:             else
7:                 currentCC = currentCC / 2
8:                 phase = SEEKING
9:             end if
10:        else
11:            if compareUtility(currentCC − 2, currentCC) > 1 and compareUtility(currentCC − 2, currentCC +
   2) > 1 then
12:                currentCC = currentCC -2
13:            else if compareUtility(currentCC+2, currentCC) > 1 and compareUtility(currentCC+2, currentCC−
   2) > 1 then
14:                currentCC = currentCC + 2
15:            else
16:                phase=FINISHED
17:            end if
18:        end if
19:        checkIfTransfersNeeded(phase)
20:    end while
21: end function
```

ING phases, we measure throughput of different concurrency values and compare them to make decision on which direction to move next. Comparison of two concurrency values depends on throughput and the value of concurrency as shown in Equation 3.1. cc refers to the value of concurrency and $thr(cc)$ refers to throughput when concurrency level cc is used. Throughput of a concurrency value is found by running transfer in that concurrency level for a certain amount of time. We observed that 15 seconds is sufficient in all cases to correctly measure the throughput for a concurrency level. Numerator in γ's calculation measures throughput increase ratio and denominator measures expected improvement ratio when concurrency is increased from cc_1 to cc_2 where $cc_2 > cc_1$. Expected improvement ratio is based on the results obtained in Figure 3.2 such that we expect to see minimum 10% increase in throughput, though expectation is higher for smaller concurrency values. For example, when comparing concurrency values 2 to 4, the expected improvement ratio would be 20% whereas it'd be 0.12% when values 10 and 12 are compared. After each iteration, it checks if it needs to measure throughput of any concurrency level to calculate γ (line 19. In the FAST_INCREASE phase, concurrency value is double as long as throughput increase ratio is greater than expected (e.g. $\gamma \geq 1$). When throughput increase

becomes smaller than expectation (e.g. $\gamma < 1$) or maximum concurrency value (aka 32) is reached, then it switches to SEEKING phase. In the SEEKING phase, we compare three concurrency values, currentCC-2, currentCC and currentCC + 2, to determine which direction to go. If smaller concurrency returns the highest utility (line 11 in Algorithm 3), then it will reduce current concurrency value and continue seeking. Similarly, if larger concurrency value obtains the highest utility, then SEEKING phase will continue in upward direction. Otherwise, it will keep the concurrency value in the current point and will finish the searching by switching to FINISHED phase.

3.3 Evaluation Results

We integrated our real time optimization method to Multi-Cluster (MC) scheduling algorithm such that it will use heuristic to estimate the value of pipelining and parallelism and use real time optimization to estimate the value of concurrency. We compared it against HARP [24] and Heuristic [5] in three networks as shown in Figure 3.3. All three methods use MC to transfer multiple clusters simultaneously. Heuristic differs from real time solution in the way it determines the value concurrency. Heuristic uses Algorithm 2 to determine the value of concurrency while real time optimization uses Algorithm 3. That is, as opposed to Heuristic which finds the value of concurrency based on network and dataset settings, this method determines it in real time by comparing different values. HARP estimates all three parameters based on models it derives using historical data and real time sample transfers.

Real time optimization method estimates the value of concurrency to be between 14 and 16 for BlueWaters-Stampede, 8 and 10 for StampedeComet, and 12 and 14 for SuperMic-Bridges networks which are very close to the peak points in Figure 3.2. On the other hand, throughput values are 10-20% less than corresponding results in Figure 3.2. This is due to overhead of search phase during which several smaller values of concurrency is tested

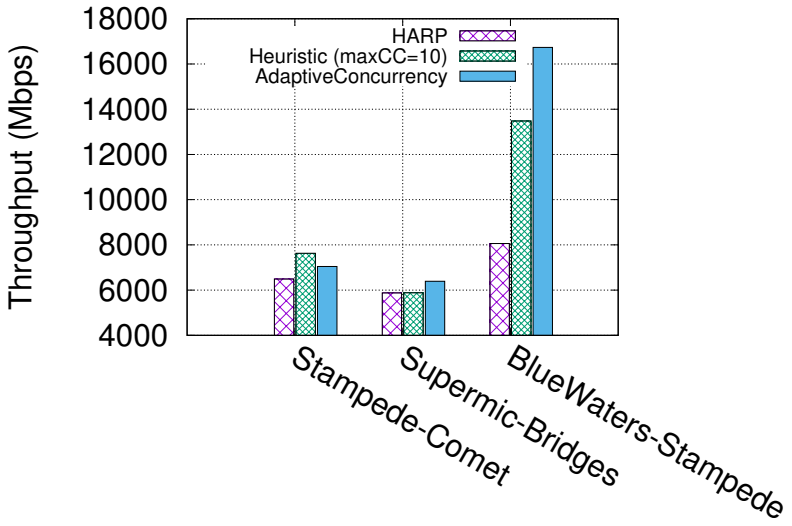


Figure 3.3: Comparison of HARP, MC-Heuristic, and MC-AdaptiveConcurrency(Real Time) algorithms.

for 15 seconds each before optimal concurrency is discovered. Real time optimization method outperforms HARP and Heuristic by 107% and 24% in BlueWaters-Stampede experiments. HARP falls short in estimating optimal parameters due to the fact that historical data only contains information for 10 Gbps networks whereas BlueWaters and Stampede is connected by 3x10 Gbps links which requires concurrency to be larger than 15 in order to reach more than 20 Gbps speed. On the other hand, Heuristic obtains 13.5 Gbps throughput when $maxCC$ is set to 10. Heuristic would perform better in higher $maxCC$ values but user needs to know what value to set which requires domain expertise. Moreover, it outperforms HARP and Heuristic slightly in SuperMic-Bridges network in which throughput improvement ratio fall below 10% after concurrency value 14. HARP estimates concurrency value to be around 11, so it performs fairly well. Similarly, Heuristic with $maxCC = 10$ is also able to achieve close to real time method. Again, while real time method is able to predict sweet spot well, its gain over others is limited due to delay in search phase. Finally, in Stampede-Comet network, Heuristic outperforms HARP and real time optimization since throughput curve breaks at around concurrency value 10-11 which

Metrics	BlueWaters-Stampede	Stampede-Comet	SuperMIC-Bridges
Throughput (Mbps)	16703	7046	6390
Concurrency value range	14-16	8-10	12-14

Table 3.1: Performance results when real time optimization algorithm is used.

is similar to $maxCC$ value. As a result, HARP performs well if historical data has similar entries of test network and Heuristic performs well as long as $maxCC$ is set properly. However, proposed real time method is able to detect sweet spot for concurrency without requiring historical data or user input. We believe this method is a step in right direction to optimize big data transfers due to being able to operate without any past information or user input.

CHAPTER 4
TIME SERIES ANALYSIS FOR HIGH PERFORMANCE SAMPLE TRANSFERS

We aim to find a good solution to estimate the average transfer throughput of a particular network quickly and accurately in the runtime. To achieve this goal, we experimented with various time-series and modelling based solutions. A time series is a sequence of time related data such that each data corresponds to a point in time. The data should be collected by regular time intervals. It is used in areas like statistics, finance, signal processing and pattern recognition. Using whole time series or part of it, time series analysis tries to extract some useful information for modeling and estimation purposes. Time series may be any data which is collected with corresponding time value for each individual record. It may be weekly (e.g. project development), daily (e.g. weather report), hourly (e.g. server usage) data or any periodic data divided within time into discrete parts. For our domain it is the throughput data divided into chosen time periods. We used one second periods for main evaluation between models. However, we also tried to change this period to lower values like hundred milliseconds to see its effect on modeling and estimation. Since only throughput data is used for analysis we implemented univariate time series analysis. It is an advantage over other methods which additionally use system parameters to predict throughput values. They basically cannot apply these univariate time series analysis techniques. Each model related to time series will be elaborated further below. The models which we evaluate in this work are:

- Negative polynomial model
- Auto Regressive (AR) model
- Auto Regressive Moving Average (ARMA) Model

- Auto Regressive Integrated Moving Average (ARIMA) model
- Adaptive Sampling
- Fixed data size model by Yildirim et al. [65]

4.1 Evaluated Models

4.1.1 Negative Polynomial Model

The negative polynomial model leverage TCP slow start behavior in which throughput of a transfer starts with small values and quickly converges to available network bandwidth. Thus, it relates transfer time to transfer throughput as shown in Equation 4.1. In the model, X_t refers to the throughput, t refers to the time since start of transfer, a and b are coefficients. Non-linear least squared method is used to calculate the value of a and b. Instead of deriving one model for all networks and transfers using historical data, we solve the equation (aka finding the a and b values) in the run-time for each transfer as transfers exhibit unique behavior based on network settings and dataset characteristics. Let's say we started transfer and observed instantaneous throughput 250, 1250, 1980 Mbps in the first three seconds, then non-linear least square method is used to estimate the coefficient values that minimizes the difference between estimated and actual throughput values. Figure 4.1 visualizes the negative polynomial model with respect to actual transfer throughput. The model is derived based on first four instant throughput data and consequent values are estimated by plugging time value in Equation 4.1 after finding the coefficient values. As it can be seen in the figure that the negative polynomial function can accurately estimate average throughput value when transfer throughput follows a predictable and stable pattern.

$$X_t = a - \frac{b}{t^2} \tag{4.1}$$

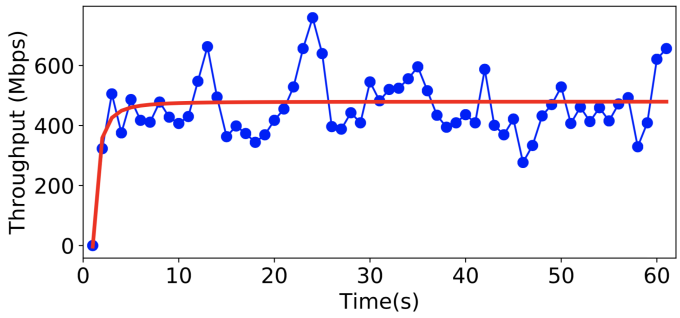


Figure 4.1: Negative polynomial function

4.1.2 Autoregressive (AR) Model

In Autoregressive model, observations from previous time steps used as input to predict the value at the next time step as shown in Equation 4.2. X_t is the predicted value, c is a constant, and φ_p is the coefficient for previous values of series. So, as it can be seen from this equation each previous value has an effect on the prediction with respect to its corresponding coefficient. Since time series data can evolve over time and may show different behaviors in different time periods, AR model only considers recent previous data to quickly adapt to changing conditions. In Equation 4.2, it considers last p data points (i.e., throughput results) when estimating the value of current time, X_t. In case of estimating expected transfer throughput, instant throughput values from first few seconds are given as input to AR to generate c and φ_i which are then used to predict i^{th}'s second throughput.

AR might be a good fit to sampling problem since it can capture throughput behavior of TCP flows. TCP uses packet sent in previous steps to calculate the number of packets to send in next step (true even in the case of packet loss events), so AR can correctly capture TCP window size regulation process when p is set properly. In addition, it also performs well when throughput results exhibit predictable stable fluctuations by means of error factor, ε_t.

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t. \tag{4.2}$$

4.1.3 Autoregressive Moving Average (ARMA) Model

Autoregressive Moving Average (ARMA) model is composed of two parts as autoregression (AR) and moving average (MA). In the equation 4.3, the first part with X_t belongs to AR and the second part with ε_{t-i} belongs to MA. The AR part makes regression on past values of throughput and the MA part constructs an error term using previous error values which represents noise in the data. Compared to AR, ARMA is more robust to recent fluctuations in data due to its feature of capturing lagged error rate. The ARMA model can understand throughput fluctuations which are caused by interference and packet loss. When TCP is exposed to a packet loss, its throughput will experience a sharp decrease followed by a recovery phase. Thus, while instant throughput may exhibit fluctuating behavior, average throughput is expected to smooth out sharp increase and decreases.

$$X_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}. \tag{4.3}$$

4.1.4 Autoregressive Integrated Moving Average (ARIMA) Model

Autoregressive Integrated Moving Average (ARIMA) model is a forecasting technique that projects the future values of a series based on AR, MA models and Integration. AR part of ARIMA is the AR model which uses p fixed previous observations to extract dependence to past observations. Integrated (I) part allows to eliminate the trend and seasonality and stabilizing the mean of the time series. Finally, MA is used to smooth out short-term fluctuations and highlight longer-term trends or cycles. Compared to AR and ARMA, ARIMA performs better when there is seasonal fluctuation in data set. In some test case we observe a distinct behavior in which throughput increase slows downs for a while and then

continues to increase in same trend. For these cases ARIMA might be a good fit since it first integrates the data and makes whole data consistent in terms of its mean and variance.

4.1.5 Adaptive Sampling

Adaptive approach relies on the assumption that instantaneous throughput will stabilize as transfer throughput converges [23]. Thus, it compares last two consecutive throughput values to determine if a transfer is converging. It stops when the last throughput value is $x\%$ closer to previous throughput value. For example, for a given time series data of instantaneous throughput 10,20,30,35,45, and $x = 8\%$, then it will stop after observing the value of 35 as the ratio between 30 and 35 is less than 8%. We evaluate its performance for 10% and 20% closeness thresholds, which means that if two points are within that percentage, then it stops and estimated the average of last two throughput reports as the throughput of sample transfer. This approach is ideal if the throughput exhibit none or little fluctuations upon convergence. However, it is susceptible to early termination when throughput of a transfer increases slowly but steadily potentially because of consecutive packet losses early in the transfer. It will also perform poorly when throughput does not stabilize significantly after it converges which can happen when transferring sets of files since instantaneous throughput may experience sharp changes when finishing one file and starting to another one due to disk I/O overhead.

4.1.6 Fixed Data Size Sampling

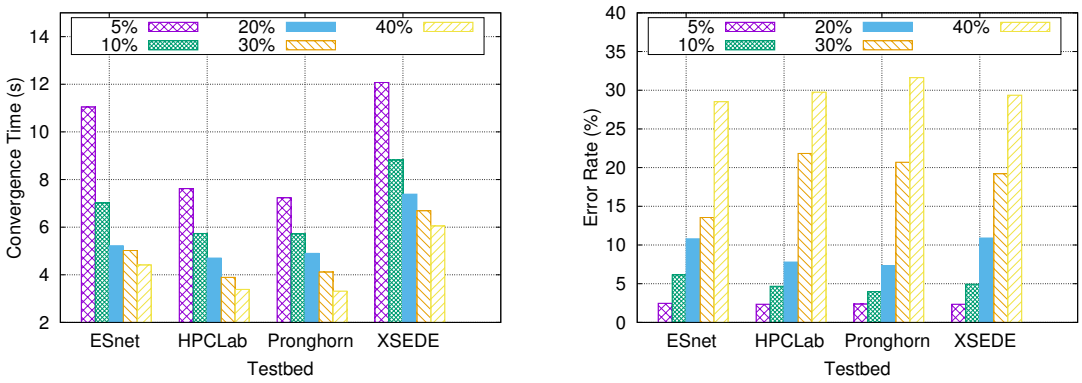
Yildirim et al. proposed a model to determine the size of dataset to use in the sample transfers [65]. Unlike the models that we discussed thus far which rely on instantaneous throughput results to estimate sample transfer throughput, it transfers a fixed portion of dataset to run sample transfers and measures time which is then used to calculate sampling throughput. To determine the size of data to transfer, it collects historical data and run

regression analysis to derive a linear model that relates sampling data size to file size and bandwidth-delay-product. On average, the model returns data size to be between 10% and 23% of original dataset size. While this is a promising step to avoid fluctuation in instantaneous throughput values, it has two major drawbacks. First, it relies on historical data to be collected in each network for each distinct datasets to accurately derive a model. Second, sample data size could be significantly large for large-scale transfers. For example, when transferring 1 TB of data, sample transfer would require 100-230 GB of data to be used which will unnecessarily take a long time to finish.

4.2 Experiments

To evaluate the described models in realistic traffic scenarios, we ran file transfers in various local and wide area networks as shown in Table 2.2 and collected transfer reports periodically. The models are then given an initial portion of each report to evaluate their accuracy in predicting actual transfer throughput for this report. For local area experiments, we used HPCLab servers at University of Nevada, Reno (UNR) and UNR campus cluster, Pronghorn. While HPCLab servers have direct attached NVMe SSDs, Pronghorn is supported by distributed file system, GPFS. XSEDE and ESnet transfers represent wide-area network conditions with 40 ms and 89 ms delay between end points. In total we ran 52,126 transfers using various file sizes (i.e, in a range of 1 MB - 100 GB) and counts. We also tested different transfer configurations by tuning application-layer patameters concurrency and parallelism. Concurrency sets the number of concurrent file transfers whereas parallelism defines the number of network connections for single file transfer. These parameters have proven to be effective in increasing transfer throughput by mitigating network and end system bottlenecks [23, 69, 70, 22]. We used custom GridFTP client to run transfers in XSEDE since data transfer nodes in XSEDE sites only support GridFTP protocol. We configured GridFTP transfers to report transfer progress in every second and logged the results

of each transfer in a separate file. We used FTP and GridFTP transfers to collect transfer logs in HPCLab, Pronghorn, and ESnet. While GridFTP report transfer throughput only once a second at most, our custom FTP protocol allowed us to fetch instantaneous throughput values in high granularity. Thus, while a majority of experiments in this paper relies on instantaneous transfer throughput collected in every one second, we evaluated sub-second data collection frequency in Section 4.3.1.



(a) Convergence Time (b) Error Rate

Figure 4.2: Evaluation of Optimal solution in terms of time and accuracy for various stopping conditions.

After running transfers and collecting instantaneous throughput periodically, we calculate average throughput of each transfer which is used to compare against the estimations of the models. Except Yildirim's model which does not process instantaneous throughput values, the accuracy of other models is measured as follows. The models are given first three data points from instantaneous throughput log file to train them. Once a model is trained, it predicts the next data point, which is compared against fourth data point from log file. If they are closer than a certain threshold, then the models is assumed to be converged and its convergence time is marked as three seconds. If the estimation is not close enough, then the models is retrained with four data points and its convergence is evaluated based on its estimation of fifth data point. The models are given more data points in instantaneous throughput until their next data point estimation matches with the actual data point

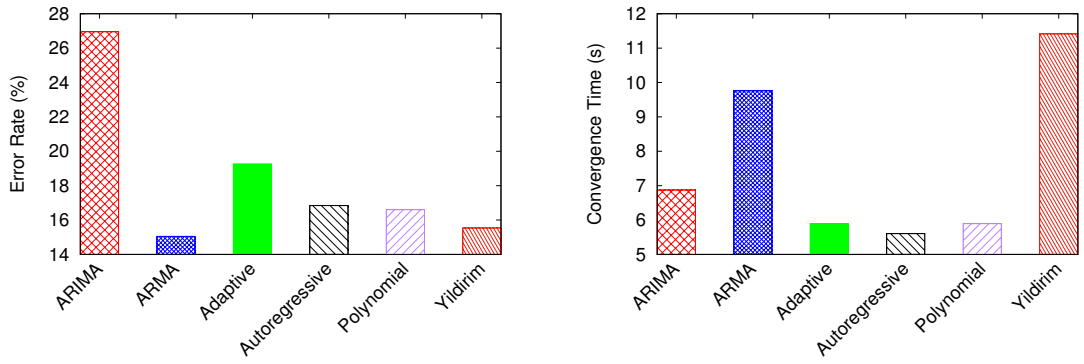
for corresponding time. Once the model is able to make accurate estimation of the next data point, the model is used to estimate average throughput, typically through estimating more data points and taking average of them. Finally, the model's average throughput estimation is compared against actual average throughput of transfer to calculate its accuracy.

Defining Optimal Solution

As shown in Figure 2.3, instantaneous throughput of transfers in different networks demonstrate difference in terms of convergence time and throughput stability. For example there might be some unexpected oscillations of throughput values in the beginning of transfer due to some background traffic and this can affect reliability of initial data in estimating the average throughput Thus, before evaluating the performance of the models in estimating sample transfer throughput, we first define *Optimal* solution that can be used to evaluate the success of models. We first calculate average throughput of transfers by dividing the datasize to transfer time. Then, the *Optimal* solution scans time-series throughput data and determines the time in which throughput is within certain range of actual average throughput. By comparing the models' performance against the optimal solution, we can see if a model performs poorly due to its design or inherent nature of the test environment. For example, if throughput of a transfer converges at 20^{th} second, then none of the models are expected to converge earlier as it would lead to inaccurate estimation of average throughput. We defined the *Optimal* solution as follows: It calculate the throughput of last four data points and compares against actual average throughput. If they are closer than a threshold, we call the last time data point as the optimal convergence time. To give an example, when instantaneous throughput of a file transfer is observed as 100, 300, 120, 610, 550, 600, 450, 400, 650, 310 Mbps, average throughput becomes 409 Mbps. When threshold is defined as 10%, the *Optimal* solution takes the average of the last four data points and stops at 5^{th} since average of 300, 120, 610, 550 becomes 395 Mbps, falls within the range of 10% error

rate of average throughput. Its error in this example becomes $\frac{|395-409|}{409} = 3.4\%$.

We evaluated various threshold cases in Figure 4.2. It is clear that as the threshold decreases, it takes longer for convergence in exchange of lower error rate. It is interesting to observe that average error rate for threshold 10% is around 5% as one might expect to close to 10% error rate when stopping condition is set to 10%. However, this is due to the fact that, threshold defines the upper bound for the error rate which means error rate for the selected data point is between 0-10% which averages to 5%. Moreover, 40% threshold leads up-to 30% error rate while keeping convergence rate less than 5 seconds. On the other hand, 5% threshold lead to 12 seconds convergence time while keeping error rate less than 3%. Since, we want to find a model that can estimate sample transfer throughput within a reasonable time and error rate values, we used 10% threshold to compare against the models in the next section.



(a) Error Rate

(b) Convergence Time

Figure 4.3: Convergence time and error rate comparison of algorithms for all network results when 10% stopping threshold is used except for fixed data size approach [65]

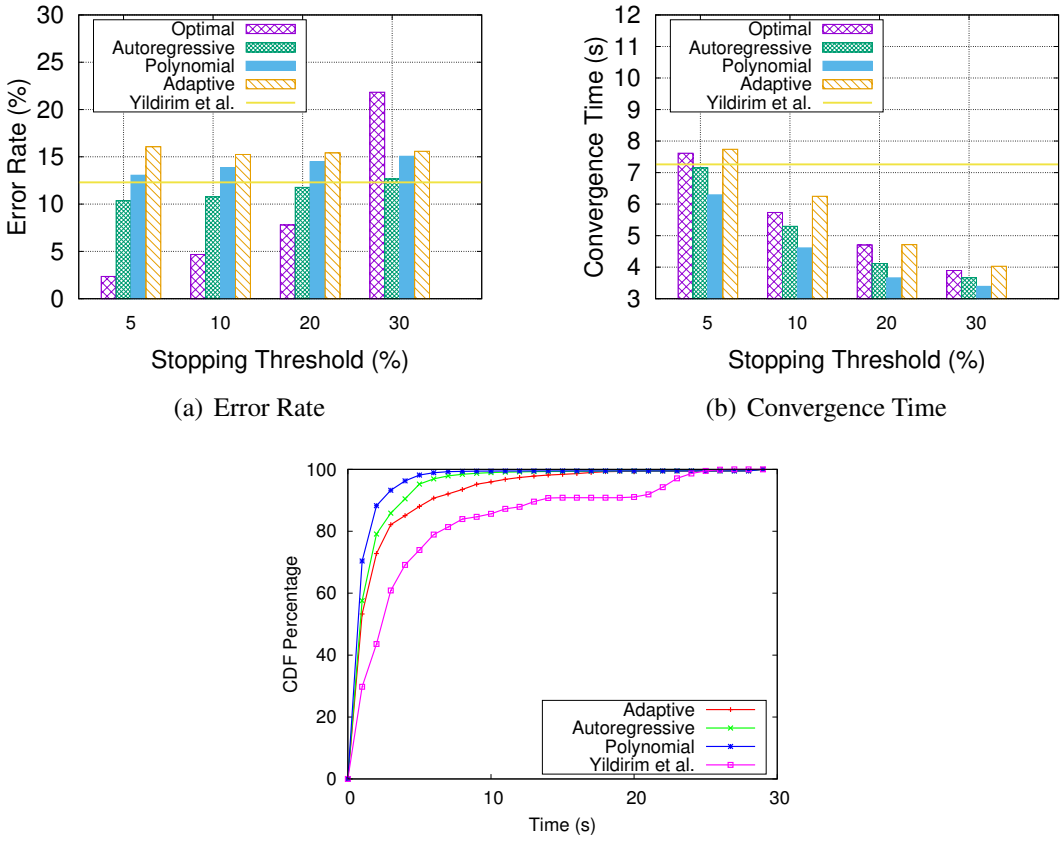
4.3 Evaluation

We tested models with both GridFTP and FTP transfers. For GridFTP transfers a java code used which utilizes libraries provided by Globus [71]. Regular FTP is done by using a

simple code written in java. For GridFTP transfer we used three different pairs; Stampede-Comet, Dtn0-Dtn1, Bridges-Stampede2. Pairs are written such that former is sender and latter is the receiver of the transfer. For each pair, several different concurrency(CC) and parallelism(P) values were set and more than a thousand transfers were made for each unique setting. Other parameters like pipelining, buffer size were kept constant within each pair.

FTP transfers are made on two different pairs. First pair is ESnet2-ESnet1 and second is Dtn1-Dtn0. One parameter was concurrency value which we set to 1, 2, 4, 8, 16 for each pair. For each pair and concurrency setting 500 transfers were done. Other parameters like parallelism, pipelining and buffer size were used with their default value for each pair.

In this section, we evaluate the performance of different models in terms of convergence time and estimation accuracy. Convergence time is determined by setting a threshold for each model as a stopping condition. We define stopping condition for each algorithm as follows. Adaptive sampling model stops when last two consecutive throughput values are close than certain percentage, thus threshold defined the percentage of closeness. For example, 5% stopping condition for Adaptive model will stop sample transfer when it observes two consecutive throughput values that are within 5% range of each other. Negative Polynomial model (shown as Polynomial in the figures) starts to read the transfer logs one by one and trains its model using least square regression. Then, the derived model is used to estimate next data point. Then, the estimation is compared against the actual data point from throughput data and it stops if estimation and actual is within 5% range. Similarly, Autoregressive model reads instantaneous throughput data one by one and trains the model. Then, the model is used to predict the next data point to compare against actual data for corresponding time value. If the prediction is 5% close to actual value, then it stops and marks the sample transfer time as completed. In the case of Yildirim's model, we did not define stopping condition since it transfers fixed data size to predict sample transfer throughput,



(a) Error Rate

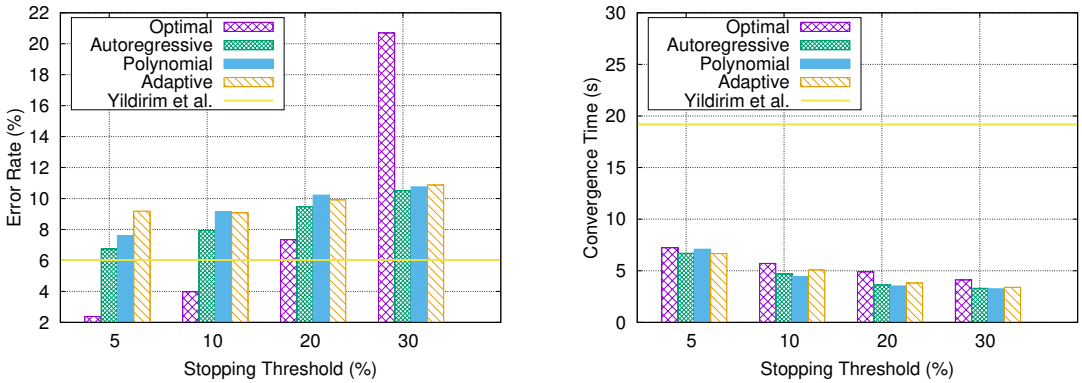
(b) Convergence Time

(c) CDF Time

Figure 4.4: Performance comparison of algorithms in HPCLab network transfers. Autoregressive model keeps the error rate below 12% and convergence time below 7 seconds.

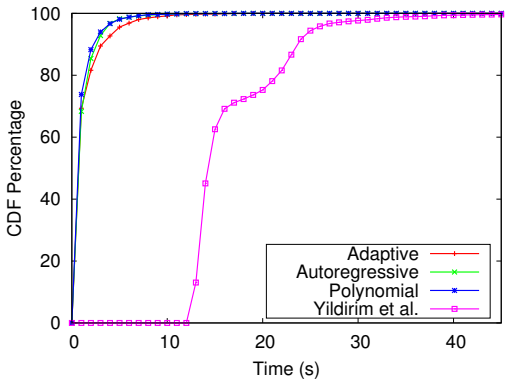
thus its error rate and accuracy values are fixed across different threshold value. Finally, we also added result from optimal solution as given in Figure 4.2 to understand how close each algorithm to optimal solution.

Figure 4.3 shows the performance comparison of the models when evaluated against all transfer logs collected in all networks. It is clear that ARIMA and Yildirim's model causes up-to 40% higher error rate without no significant improvement in convergence time. While ARMA can obtain around 10% lower error rate compared to AutoRegressive and Polynomial models, its convergence time is almost 50% higher than the other two models. AutoRegressive and Polynomial models appear to yield the best performance when



(a) Error Rate

(b) Convergence Time

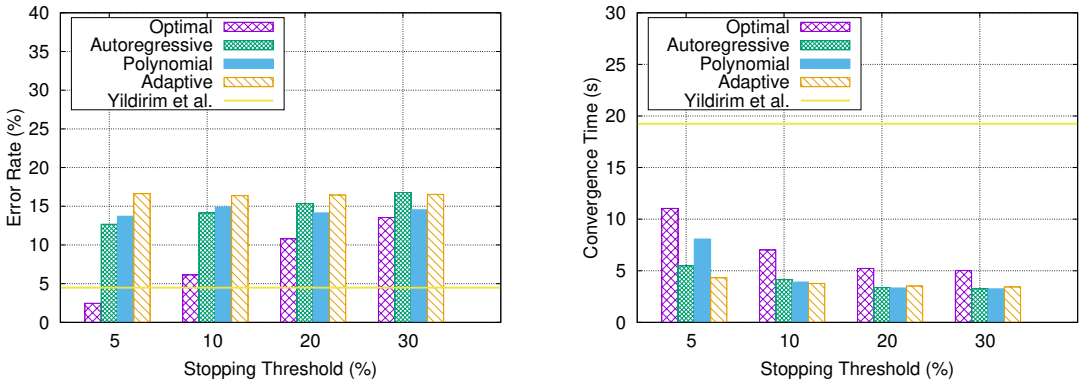


(c) CDF Stopping Time

Figure 4.5: Performance comparison of algorithms in Pronghorn campus cluster. Autoregressive model yields the lower error rate compared to Adaptive and Polynomial models with less than 10% in all thresholds values.

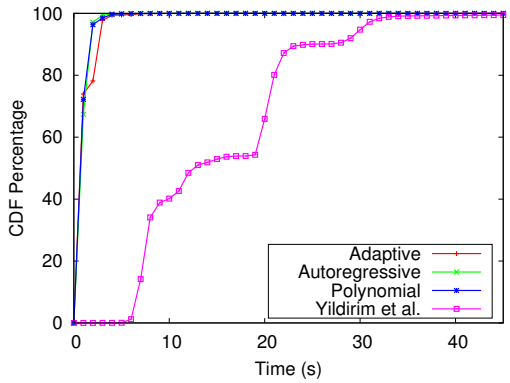
both time and error rate considered considered at the same time. Hence, in the following results, we omit ARMA and ARIMA and compare the other solutions against each other and *Optimal* solution.

Figure 4.4 shows error rate and convergence time of models in HPCLab testbed. Autoregressive model yields the lowest error rate for all threshold values. As the stopping condition increases from 5% to 30%, error rate for Autoregressive increase slightly due to fairly stable transfer throughput in this network. On the other hand, average convergence time of all transfers in HPCLab network decreases from around 7 seconds to around 4 seconds since the model can find a throughput value to satisfy stopping condition ear-



(a) Error Rate

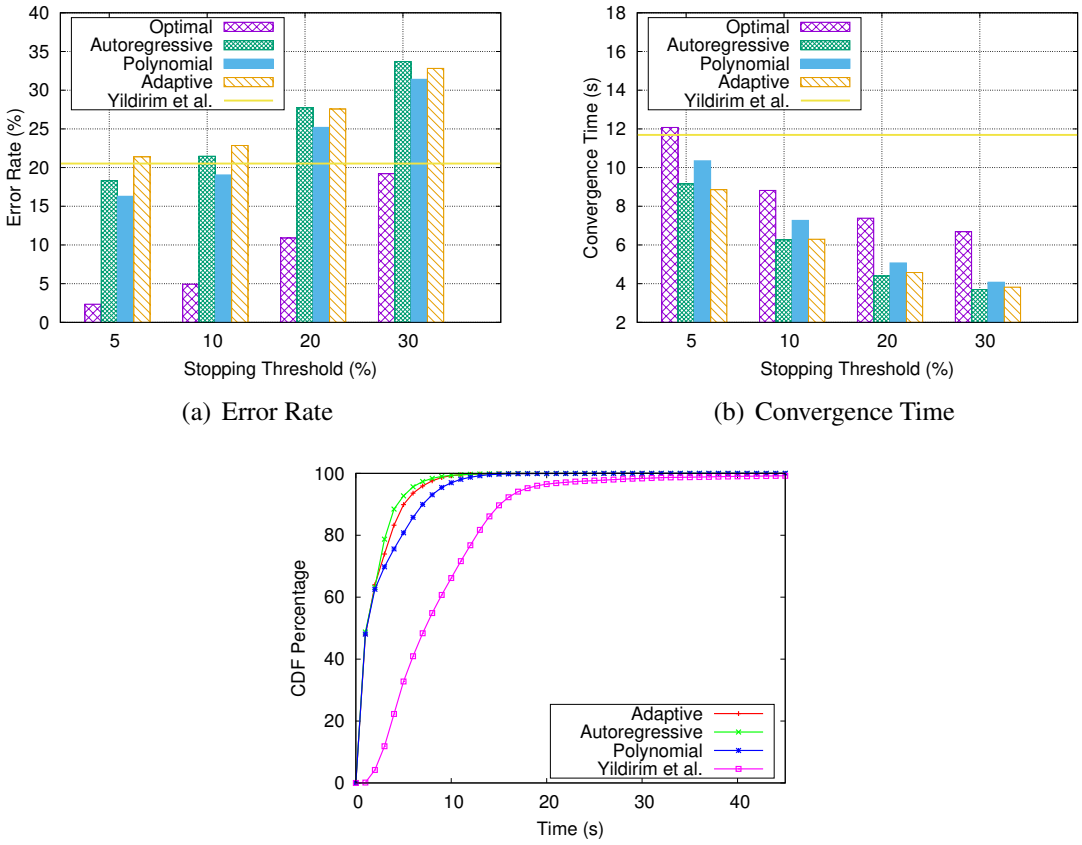
(b) Stopping Time



(c) CDF Convergence Time

Figure 4.6: Performance comparison of algorithms in ESnet network transfers. While polynomial model performs similar to Autoregressive model is most cases, its convergence time for 5% threshold is 25% worse.

lier. These results indicate that, throughput of transfers in HPCLab experiments converges quickly and exhibit fair amount of fluctuations upon the convergence. Consequently, while large stopping threshold values yield up-to 80% quick decision with only 10% worse error rate compared to low stopping condition cases. On the other hand, Yildirim's model achieves 12.5% error rate wirh 7.8s convergence time. Compared to other models, its accuracy is mostly better but convergence time is nearly twice large than others for increased stopping threshold case. It is also important to note that, Yildirim's approach requires an up-front work to derive a model that can estimate optimal sampling size. Adaptive model performs the worst of all in all stopping thresholds which can be attributed to its inability



(a) Error Rate

(b) Convergence Time

(c) CDF Stopping Time

Figure 4.7: Performance comparison of algorithms in XSEDE network. As opposed to other testbeds, XSEDE causes significantly higher error rates due to its shared nature of end system and network resources.

to leverage from complete history as it only considers last two data points. Finally, polynomial model has the fastest convergence time but its error rate is higher than Autoregressive model. Figure 4.4(c) shows cumulative distribution function for convergence time with 20% convergence rate.

Figure 4.5 shows the evaluation results for Pronghorn testbed. Similar to HPCLab network, Pronghorn experiments are conducted between two servers in same local area network. While Yildirim's model has the lowest error rate with 6%, its convergence time is nearly 4x higher than other models. Again, Autoregressive model yields the lower error

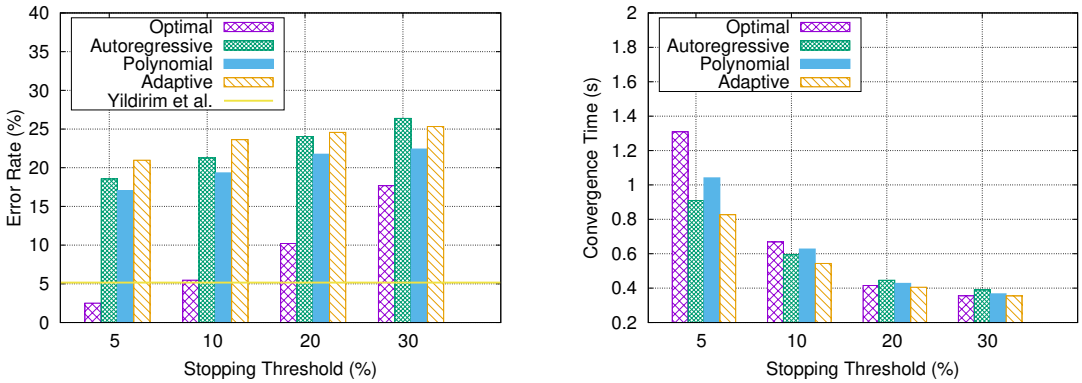
rate compared to Adaptive and Polynomial models with less than 10% in all thresholds values. Its convergence time is also similar to Polynomial model which has the lowest convergence time in all threshold values except 5%. Adaptive model again falls behind of Autoregressive and Polynomial both in convergence time and error rate.

Figure 4.6 shows the results for ESnet testbed where source and destination end points are connected with 100 Gbps bandwidth and 89 ms RTT. Yildirim's model achieves as low as 5% error rate which is close to optimal solution. In exchange, it leads to significantly higher time to converge as shown in Figure 4.6(b). Polynomial and Autoregressive models perform similar in most cases with less than 15% error rate and less than 5 seconds convergence time. However, Polynomial model takes 25% more time to converge in 5% threshold case. On the orher hand, Adaptive approach achieves less than 4 seconds convergence time at all threshold conditions but yields 20-25% higher error rate compared to Autoregressive and Polynomial models. CDF of convergence time again reveals that Yildirim's model leads to significantly high convergence time. Autoregressive, Polynomial, and Adaptive models, however, perform similar with Polynomial model converging slightly better.

Th results for XSEDE experiments are given in Figure 4.7. As opposed to other testbeds, XSEDE causes significantly higher error rates due to its shared nature of end system and network resources. While error rates for HPCLab, Pronghorn, and ESnet for Autoregressive were below 15% for all threshold levels, it exceeds to 30% in 30% threshold case in XSEDE. Polynomial model yields 10-20% lower error rate compared to Autoregressive and Adaptive models. In return, its convergence time is 10-15% higher. Autoregressive model yields the lowest convergence time in all threshold values. In overall, Autoregressive model performs consistent across all testbed with reasonably well accuracy and convergence time values. Moreover, Although Yildirim's model achieves low error rate in all networks, its convergence time is prohibitively high, so it cannot be used for real-time optimization. While Polynomial model can also offer comparable performance

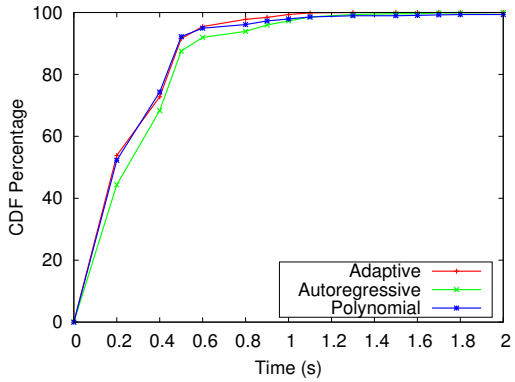
in most cases, its error rate can be up-to 20% worse than Autoregressive.

4.3.1 The Impact of Data Collection Frequency



(a) Error Rate

(b) Stopping Time



(c) CDF Stopping Time

Figure 4.8: Collecting throughput reports at higher granularity can help to significantly reduce sample transfer time.

Experiments we considered so far were based on instantaneous throughput data that is collected once in every second. In this section, we investigate the impact of collecting throughput results in sub-second intervals in an attempt to achieve faster convergence time which is critical to be able to run many sample transfers in a timely manner. While GridFTP servers do not support populating instantaneous throughput values in sub-second intervals, we configured our custom FTP client report transfer throughput in every 100ms in HPCLab

and Pronghorn networks. Then, we evaluate the models and to calculate the convergence time and error rate as shown in Figure 4.8(c). Yildirim's model is not shown in the average convergence time and CDF of convergence time figures (Figure 4.8(b) and 4.8(c)) as it takes nearly 10x more time than other models since it does not rely on instantaneous throughput and transfers large portion of dataset to run sample transfers.

Figure 4.8(b) shows that all models converge in less than a second. On the other hand, this leads to increased error rate. While Autoregressive model achieves less than 12% error rate for HPCLab and Pronghorn networks as shown in Figure 4.4 and 4.5, its error rate reaches to 20% when tested with more granular throughput data. However, this can be a reasonable trade-off when compared to 4-6x reduction in convergence time.

CHAPTER 5
CONCLUSION AND FUTURE WORK

Driven by advancements in instrument technologies and computing power, many scientific applications have started to generate massive volumes of data. Moreover, the collaborative and distributed nature of science requires large-scale data movement across geographically dispersed institutions to enable data processing, storage, and sharing. Thus it is of the utmost important to effectively utilize available network and end system resources to maximize transfer throughput between research and educational institutions around the globe. In this thesis, we focused on deriving a real time approach to tune transfer settings in addition to optimization of sample transfers to maximize the benefit of real-time optimization.

In the first work, we proposed an adaptive method to tune concurrency value in the real time. Optimal concurrency value is the one which leads high throughput while not causing too much burden on end systems and network. We defined a utility function to compare different concurrency values and determine the one with higher benefit in overall. Real time tuning method is compared against the state-of-the-art transfer tuning solutions HARP and observed up to 124% improvement in transfer throughput. On the other hand, it requires around fifteen seconds to assess throughput value of a given concurrency using sample transfers. While fifteen seconds may not seem too much, it would add up to a significant time duration when evaluating several concurrency values.

In the second work, we proposed and evaluated various models to estimate throughput of sample transfers to alleviate sampling overhead for real-time transfer optimization. Beside being useful for enhancing our work this can be used in any application which requires to make average throughput estimation in limited time. The results indicate that time-series analysis using Autoregressive model can achieve less than 12% error rate in most cases with

less than 5 seconds convergence time. However, the error rate and convergence time increase in shared networks due to resource interference at the storage, server, and network levels. Moreover, we found that collecting instantaneous throughput results more often than once a second can help to lower convergence time over 6 times while deteriorating error rate by 5-10%.

As future work, effects of already observed parameters and other different parameters will be studied more in detail. It might be done either by considering more parameters that affect throughput or controlling the environment more and understanding effect of one parameter better. Additional parameters to consider can be related to both network and end hosts such as disk read speed, file system type or busy times of network. These would shed a light on reasons for different behaviors that happens in seemingly similar settings by our current measure. Second direction is about controlling the environment to keep external factors under control. When external factors are controlled, correlations can be understood properly and better models can be created. Moreover, we aim to investigate the models in controlled congestion environment to find out how they are affected as background traffic intensifies.

REFERENCES

[1] R. J. T. Klein, R. J. Nicholls, and F. Thomalla, "Resilience to natural hazards: How useful is this concept?" *Global Environmental Change Part B: Environmental Hazards*, vol. 5, no. 1-2, pp. 35 –45, 2003.

[2] J. Kiehl, J. J. Hack, G. B. Bonan, B. A. Boville, D. L. Williamson, and P. J. Rasch, "The national center for atmospheric research community climate model: CCM3," *Journal of Climate*, vol. 11:6, pp. 1131–1149, 1998.

[3] CMS, *The US Compact Muon Solenoid Project*, http://uscms.fnal.gov/.

[4] E. H. T. Project, https://eventhorizontelescope.org/.

[5] E. Arslan, B. Ross, and T. Kosar, "Dynamic protocol tuning algorithms for high performance data transfers," in *Proceedings of Euro-Par'13*, ser. Euro-Par'13, Aachen, Germany: Springer-Verlag, 2013, pp. 725–736, ISBN: 978-3-642-40046-9.

[6] E. Arslan, B. A. Pehlivan, and T. Kosar, "Big data transfer optimization through adaptive parameter tuning," *Journal of Parallel and Distributed Computing*, vol. 120, pp. 89–100, 2018.

[7] H. Sapkota, B. Pehlivan, and E. Arslan, "Time series analysis for efficient sample transfers," in *Proceedings of the 2nd Workshop on Systems and Network Telemetryand Analytics*, ACM, 2019.

[8] L. S. Brakmo and L. L. Peterson, "Tcp vegas: End to end congestion avoidance on a global internet," *IEEE Journal on selected Areas in communications*, vol. 13, no. 8, pp. 1465–1480, 1995.

[9] R. Karrer, J Park, and J. Kim, "Tcp-rome:performance and fairness in parallel downloads for web and real time multimedia streaming applications," in *In Technical Report, Deutsche Telekom Laboratories*, 2006.

[10] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 329–342.

[11] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "{pcc} vivace: Online-learning congestion control," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18))*, 2018, pp. 343–356.

[12] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Queue*, vol. 14, no. 5, p. 50, 2016.

[13] Y. Gu and R. L. Grossman, "Udt: Udp-based data transfer for high-speed wide area networks," *Computer Networks*, vol. 51, no. 7, pp. 1777–1799, 2007.

[14] C. Raiciu, C. Pluntke, S. Barre, A. Greenhalgh, D. Wischik, and M. Handley, "Data center networking with multipath tcp," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX, Monterey, California: ACM, 2010, 10:1–10:6, ISBN: 978-1-4503-0409-2.

[15] G. Khanna, U. Catalyurek, T. Kurc, R. Kettimuthu, P. Sadayappan, I. Foster, and J. Saltz, "Using overlays for efficient data transfer over shared wide-area networks," in *Proceedings of SC*, Piscataway, NJ, USA, 2008, pp. 1–12.

[16] N. Freed, *SMTP service extension for command pipelining*, http://tools.ietf.org/html/rfc2920.

[17] T. J. Hacker, B. D. Noble, and B. D. Atley, "Adaptive data block scheduling for parallel streams," in *Proceedings of HPDC '05*, ACM/IEEE, Jul. 2005, pp. 265–275.

[18] 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*, vol. 2(4), pp. 177–183, Sep. 2008.

[19] N. S. Rao, Q. Liu, S. Sen, G. Hinkel, N. Imam, I. Foster, R. Kettimuthu, B. W. Settlemyer, C. Q. Wu, and D. Yun, "Experimental analysis of file transfer rates over wide-area dedicated connections," in *IEEE 18th High Performance Computing and Communications*, IEEE, 2016, pp. 198–205.

[20] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The globus striped gridftp framework and server," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, IEEE Computer Society, 2005, p. 54.

[21] B. Allen, J. Bresnahan, L. Childers, I. Foster, G. Kandaswamy, R. Kettimuthu, J. Kordas, M. Link, S. Martin, K. Pickett, and S. Tuecke, "Software as a service for data scientists," *Communications of the ACM*, vol. 55:2, pp. 81–88, 2012.

[22] R. Kettimuthu, G. Vardoyan, G. Agrawal, and P Sadayappan, "Modeling and optimizing large-scale wide-area data transfers," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, IEEE, 2014, pp. 196–205.

[23] E. Arslan and T. Kosar, "High speed transfer optimization based on historical analysis and real-time tuning," *IEEE Transactions on Parallel and Distributed Systems*, 2018.

[24] E. Arslan, K. Guner, and T. Kosar, "Harp: Predictive transfer optimization based on historical analysis and real-time probing," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '16, Salt Lake City, Utah: IEEE Press, 2016, 25:1–25:12, ISBN: 978-1-4673-8815-3.

[25] E. Yildirim, E. Arslan, J. Kim, and T. Kosar, "Application-level optimization of big data transfers through pipelining, parallelism and concurrency," *Cloud Computing, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[26] E. Arslan and T. Kosar, *A heuristic approach to protocol tuning for high performance data transfers*, 2017. arXiv: 1708.05425 [cs.DC].

[27] N. Freed, *Smtp service extension for command pipelining*, United States, 2000.

[28] K Farkas, P Huang, B Krishnamurthy, Y Zhang, and J Padhye, "Impact of tcp variants on http performance," *Proceedings of High Speed Networking*, vol. 2, p. 13, 2002.

[29] J. Kim, E. Yildirim, and T. Kosar, "A highly-accurate and low-overhead prediction model for transfer throughput optimization," *Cluster Computing*, vol. 18, no. 1, pp. 41–59, 2015.

[30] E. Altman and D. Barman, "Parallel tcp sockets: Simple model, throughput and validation," in *Proceedings of IEEE INFOCOM*, 2006, pp. 1–12.

[31] T. J. Hacker, B. D. Noble, and B. D. Athey, "Adaptive data block scheduling for parallel tcp streams," in *Proceedings of HPDC*, 2005, pp. 265–275.

[32] D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante, "Modeling and taming parallel tcp on the wide area network," in *Proceedings of IPDPS*, 2005, 68b–68b.

[33] E. Yildirim and T. Kosar, "Network-aware end-to-end data throughput optimization," in *Proceedings of the first international workshop on Network-aware data management*, ACM, 2011, pp. 21–30.

[34] J. Bresnahan, M. Link, R. Kettimuthu, D. Fraser, and I. Foster, "Gridftp pipelining," in *Proceedings of TeraGrid*, 2007.

[35] T. Kosar and M. Livny, "Stork: Making data placement a first class citizen in the grid," in *Proceedings of ICDCS'04*, Mar. 2004, pp. 342–349.

[36] W. Liu, B. Tieman, R. Kettimuthu, and I. Foster, "A data transfer framework for large-scale science experiments," in *Proceedings of HPDC'10*, ser. HPDC '10, Chicago, Illinois: ACM, 2010, pp. 717–724, ISBN: 978-1-60558-942-8.

[37] T. Kosar, M. Balman, E. Yildirim, S. Kulasekaran, and B. Ross, "Stork data scheduler: Mitigating the data bottleneck in e-science," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 369, no. 1949, pp. 3254–3267, 2011.

[38] T. Kosar, *Data intensive distributed computing: Challenges and solutions for large-scale information management*, 2012.

[39] E. Yildirim, I. H. Suslu, and T. Kosar, "Which network measurement tool is right for you? a multidimensional comparison study," in *Proceedings of Grid'08*, IEEE, 2008, pp. 266–275.

[40] M. S.Q. Z. Nine, K. Guner, and T. Kosar, "Hysteresis-based optimization of data transfer throughput," in *Proceedings of NDM'15*, ser. NDM '15, Austin, Texas: ACM, 2015, 5:1–5:9, ISBN: 978-1-4503-4037-3.

[41] E. Yildirim, E. Arslan, J. Kim, and T. Kosar, "Application-level optimization of big data transfers through pipelining, parallelism and concurrency," *IEEE Trans. Cloud Computing*, vol. 4, no. 1, pp. 63–75, Jan. 2016.

[42] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM conference on Computer and communications security*, Acm, 2007, pp. 598–609.

[43] Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multicloud storage," *IEEE transactions on parallel and distributed systems*, vol. 23, no. 12, pp. 2231–2244, 2012.

[44] C. Liu, C. Yang, X. Zhang, and J. Chen, "External integrity verification for outsourced big data in cloud and IoT: A big picture," *Future generation computer systems*, vol. 49, pp. 58–67, 2015.

[45] P. Maniatis, M. Roussopoulos, T. J. Giuli, D. S. Rosenthal, and M. Baker, "The LOCKSS peer-to-peer digital preservation system," *ACM Transactions on Computer Systems (TOCS)*, vol. 23, no. 1, pp. 2–50, 2005.

[46] M. Vigil, J. Buchmann, D. Cabarcas, C. Weinert, and A. Wiesmaier, "Integrity, authenticity, non-repudiation, and proof of existence for long-term archiving: A survey," *Computers & Security*, vol. 50, pp. 16–32, 2015.

[47] A. Ma, C. Dragga, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. K. Mckusick, "Ffsck: The fast file-system checker," *ACM Transactions on Storage (TOS)*, vol. 10, no. 1, p. 2, 2014.

[48] Y. Zhang, D. S. Myers, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Zettabyte reliability with flexible end-to-end data integrity," in *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*, IEEE, 2013, pp. 1–14.

[49] Y. Zhang, A. Rajimwale, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "End-to-end data integrity for file systems: A ZFS case study.," in *FAST*, 2010, pp. 29–42.

[50] M. U. Arshad, A. Kundu, E. Bertino, A. Ghafoor, and C. Kundu, "Efficient and scalable integrity verification of data and query results for graph databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 5, pp. 866–879, 2018.

[51] R. Hasan, R. Sion, and M. Winslett, "The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance.," in *FAST*, vol. 9, 2009, pp. 1–14.

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

[53] *Globus*, https://www.globus.org/.

[54] E. Arslan and A. Alhussen, "A low-overhead integrity verification for big data transfers," in *2018 IEEE International Conference on Big Data (Big Data)*, IEEE, 2018, pp. 4227–4236.

[55] B. Charyyev, A. Alhussen, H. Sapkota, E. Pouyou, M. Gunes, and E. Arslan, "Towards securing data transfers against silent data corruption," in *IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing*, IEEE/ACM, 2019.

[56] XSEDE, *Extreme Science and Engineering Discovery Environment*, http://www.xsede.org/, 2018.

[57] I. Alan, E. Arslan, and T. Kosar, "Energy-performance trade-offs in data transfer tuning at the end-systems," *Sustainable Computing: Informatics and Systems*, vol.

4, no. 4, pp. 318 –329, 2014, Special Issue on Energy Aware Resource Management and Scheduling (EARMS).

[58] I. Alan, E. Arslan, and T. Kosar, "Energy-aware data transfer algorithms," in *Proceedings of SC'15*, ser. SC '15, Austin, Texas: ACM, 2015, 44:1–44:12, ISBN: 978-1-4503-3723-6.

[59] E. Arslan, K. Guner, and T. Kosar, "Harp: Predictive transfer optimization based on historical analysis and real-time probing," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '16, Salt Lake City, Utah: IEEE Press, 2016, 25:1–25:12, ISBN: 978-1-4673-8815-3.

[60] D. Yun, C. Q. Wu, N. S. Rao, Q. Liu, R. Kettimuthu, and E.-S. Jung, "Data transfer advisor with transport profiling optimization," in *Local Computer Networks (LCN), 2017 IEEE 42nd Conference on*, IEEE, 2017, pp. 269–277.

[61] Z. Liu, R. Kettimuthu, I. Foster, and P. H. Beckman, "Toward a smart data transfer node," *Future Generation Computer Systems*, 2018.

[62] A. Hanemann, J. W. Boote, E. L. Boyd, J. Durand, L. Kudarimoti, R. Łapacz, D. M. Swany, S. Trocha, and J. Zurawski, "Perfsonar: A service oriented architecture for multi-domain network monitoring," in *International conference on service-oriented computing*, Springer, 2005, pp. 241–254.

[63] S. Pandey and R. Buyya, "Scheduling workflow applications based on multi-source parallel data retrieval in distributed computing networks," *The Computer Journal*, vol. 55, no. 11, pp. 1288–1308, 2012.

[64] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate adaptation for adaptive http streaming," in *Proceedings of the second annual ACM conference on Multimedia systems*, ACM, 2011, pp. 169–174.

[65] E. Yildirim, J. Kim, and T. Kosar, "Modeling throughput sampling size for a cloud-hosted data scheduling and optimization service," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1795–1807, 2013.

[66] I. Alan, E. Arslan, and T. Kosar, "Energy-aware data transfer algorithms," in *Proceedings of Supercomputing*, 2015.

[67] P. Balaprakash, V. Morozov, R. Kettimuthu, K. Kumaran, and I. Foster, "Improving data transfer throughput with direct search optimization," in *Parallel Processing (ICPP), 2016 45th International Conference on*, IEEE, 2016, pp. 248–257.

[68] R. P. Karrer, "Tcp prediction for adaptive applications," in *32nd IEEE Conference on Local Computer Networks (LCN 2007)*, IEEE, 2007, pp. 989–996.

[69] E. Yildirim and T. Kosar, "Network-aware end-to-end data throughput optimization," in *Proceedings of the first international workshop on Network-aware data management*, ser. NDM '11, Seattle, Washington, USA: ACM, 2011, pp. 21–30, ISBN: 978-1-4503-1132-8.

[70] E. Yildirim, D. Yin, and T. Kosar, "Prediction of optimal parallelism level in wide area data transfers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 12, pp. 2033–2045, 2011.

[71] JGlobus, https://github.com/jglobus/jglobus, 2019.