Time Series Analysis for Efficient Sample Transfers

Hemanta Sapkota
Computer Science and Engineering
Reno, Nevada
hsapkota@nevada.unr.edu

Bahadir A. Pehlivan
Computer Science and Engineering
Reno, Nevada
bpehlivan@nevada.unr.edu

Engin Arslan
Computer Science and Engineering
Reno, Nevada
earslan@unr.edu

ABSTRACT

Real-time transfer optimization approaches offer promising solutions as they can discover optimal transfer configuration in the runtime without requiring an upfront work or making assumptions about underlying system architectures. On the other hand, existing implementations suffer from slow convergence speed due to running many sample transfers with suboptimal configurations. In this work, we evaluate time-series models to minimize the impact of sample transfers with suboptimal configurations by shortening the transfer duration without degrading the accuracy. 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 which is up-to 4x improvement over the state-of-the-art solution. We also realized that while the most common transfer applications report transfer throughput at most once a second, decreasing the reporting interval is the key to further reduce the impact of sample transfers by quickly determining their performance.

ACM Reference Format:
Hemanta Sapkota, Bahadir A. Pehlivan, and Engin Arslan. 2019. Time Series Analysis for Efficient Sample Transfers. In *Systems and Network Telemetry and Analytics (SNTA'19), June 25, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3322798.3329256

1 INTRODUCTION

Large scientific experiments such as environmental and coastal hazard prediction [17], climate modeling [15], and high-energy physics simulations [9] 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. Even though the existing high speed networks with up-to 100 Gbps network bandwidth have been established, many users still experience difficulty reaching the theoretical maximum throughput, causing underutilization of resources [7]. A common way to address low-performance problems is tuning the application level transfer configurations such as pipelining [10], parallelism [11], concurrency [18], buffer size [13], block size [23], and striping [2]. While significant performance gains can be achieved by tuning these parameters [6, 10, 23], the optimal configuration 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 transport protocol). Thus, finding the best parameter combination is a challenging task due to large search space and prohibitive cost of exhaustive profiling.

There have been several attempts to tune some of these application-layer parameters to maximize transfer throughput using heuristic [3, 6, 7], supervised [14, 21], semi-supervised [4, 5], and unsupervised [20, 23, 24, 28] 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 are favored 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 [12], workflow scheduling [22], and adaptive video streaming [19].

Current solutions to conduct sample transfers include fixed data size [25], fixed-time duration [1], and adaptive approach [5]. In the fixed data size approach, a pre-calculated amount of an original dataset (e.g., 10%) is used to run sample transfers, however it requires an up-front work to determine the optimal data size which may not be feasible in every network [25]. 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. Adaptive sampling initiates whole dataset transfer and keeps track of throughput periodically (e.g., once a second). It assumes that transfer throughput is converged to a value when throughput ratio of two consecutive intervals is closer than a defined threshold. Finally, it stops the transfer and takes average of last two throughput results as the throughput of the sample transfer. Among them, adaptive approach promises fast convergence with the

SNTA'19, June 25, 2019, Phoenix, AZ, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6761-5/19/06...$15.00
https://doi.org/10.1145/3322798.3329256

highest accuracy, however we found that it can fail to converge when transfer throughput exhibit fluctuations due to network instability or inaccurate throughput measurements.

In this paper, we propose time-series models to predict sample transfer performance in the runtime expeditiously and accurately. As opposed to previous work, we neither rely on historical data nor low-level TCP metrics. Instead, the models rely on real-time throughput observations to predict throughput of sample transfers upon convergence, minimizing the cost of sample transfers without sacrificing estimation accuracy. Contributions of this paper are as follows:

- We introduce several time-series models to process real-time throughput metrics to predict convergence throughput of sample transfers in short time duration with high estimation accuracy.
- We investigate the impact of measuring transfer throughput more frequently in attempt to reduce time to derive time-series models and estimate convergence throughput in less than a second.
- We conduct extensive experiments using two local area and two wide area networks with various dataset, network, and transfer configurations to evaluate the proposed models in wide range of scenarios.

The rest of paper is organized as follows: Section 2 explains the motivation and presents related work in the area of sample transfer optimization. Section 3 describes the proposed models as well as the state-of-the-art solutions. Section 3 explains experimental setup and Section 5 discusses the evaluation results. Finally, Section 6 concludes the paper with the summary and potential future directions.

2 MOTIVATION AND RELATED WORK

As pointed out by previous works, the throughput of a data transfer in shared networks rely on various factors including dataset characteristics, network settings, transfer configurations, and background load [5–7, 14, 16]. Hence, it is hard to predict transfer throughput of a dataset in advance. Sample transfers are used to estimate transfer throughput by probing network with a given transfer configuration and are used to identify a configurations that yields the maximum transfer throughput. The most common way to run sample transfers is to use a portion of original dataset such that sample transfers could also contribute to actual transfer. However, there is no consensus on how to schedule sample transfers as obtaining accurate and timely results is a difficult task due to unpredictable nature of resource interference.

Figure 1 demonstrates throughput fluctuations over time for several transfers in different networks. It is clear in the figure that some transfers experience sharp changes in observed throughput. In addition, transfer throughput stabilizes at different times in different networks. While one transfer can reach to maximum throughput in as short as 3 seconds, it can take up-to 20 seconds for another one. Although running sample transfers for the worst-case scenario is possible to guarantee reaching to maximum possible throughput, it would significantly increase the search time for real-time

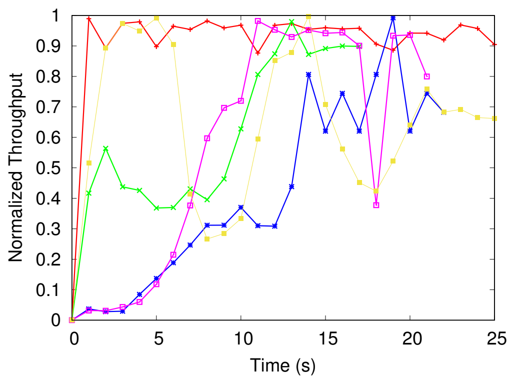


Figure 1: Convergence behavior of file transfers throughput follows distinct pattern based on network settings, dataset characteristics and background traffic.

transfer optimization algorithms that aim to find the optimal transfer configuration by running a series of sample transfers. Thus, it is nontrivial to schedule sample transfers that can capture fast and accurate results without imposing significant overhead. Researchers proposed fixed-size [25], fixed-duration [1, 8], adaptive [4], and modelling-based [25] techniques to conduct sample transfers.

Yildirim et al. developed a model to estimate percentage of dataset size to be used in sample transfers [25]. They first run extensive experiments to collect accuracy statistics for various sampling sizes and then runs regression analysis to extract the relationship between sampling size and transfer conditions 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 a 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. Moreover, real-time transfer optimization algorithms would end up transferring the most, if not all, of the dataset during search phase, turning the identification of the optimal configuration hardly useful.

In another work, throughput of sample transfers are determined by running them for a fixed time duration [1]. The goal is to evaluate different transfer configurations and determine the one with the highest throughput to energy consumption ratio. The authors concluded that 5 seconds is sufficient to predict the performance of any transfer configuration in their test networks. However, running sample transfers for a fixed time period involves sensitive tuning of the duration based on network conditions since a single value could be too short for some networks and too long for others. 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.

In a previous work [4], we proposed an adaptive sampling in which we start transferring an entire dataset and monitor throughput periodically. If throughput of two consecutive monitor intervals are closer than a defined threshold, we stop 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 does not exhibit much fluctuations upon convergence. However, our experiments showed that this assumption may not hold true in shared networks with unpredictable network and end system congestion.

3 PROPOSED MODELS

We aim to estimate the transfer throughput of sample transfers quickly and accurately in the runtime. To achieve this goal, we experiment with following time-series and regression based models:

- Negative polynomial model
- Auto Regressive (AR) model
- Auto Regressive Moving Average (ARMA) Model
- Auto Regressive Integrated Moving Average (ARIMA) model
- Adaptive Sampling by Arslan et al. [5]
- Fixed data size mode by Yildirim et al. [25]

3.1 Negative Polynomial Model

The negative polynomial model leverages 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 1. In the model, X_t refers to the throughput, t refers to the time since start of transfer, a and b are coefficients. Nonlinear 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 assume we started a transfer and observed throughput values 250, 1250, 1980 Mbps in the first three seconds, then nonlinear least square method is used to estimate the coefficient values that minimizes the difference between estimated and actual throughput values. Figure 2 visualizes the negative polynomial models with respect to actual transfer throughput. The model is derived based on first four throughput data[1]. After deriving the equation and finding corresponding a and b values, future values are estimated by plugging increasing time values in Equation 1. 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.

$$X_t = a - \frac{b}{t^2} \tag{1}$$

3.2 Autoregressive (AR) Model

In Autoregressive model uses observations from previous time steps as input to predict the value at the next time step as shown in Equation 2. X_t is a predicted value, c is a constant, and φ_p is the coefficient for the lagged variable. Since time series data can evolve over time, AR model only considers recent previous data to quickly adapt to changing conditions. In Equation 2, it considers last p data points (i.e., throughput results) when estimating the value of current time, X_t. In case of estimating sample transfer throughput estimation, 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.

[1]Unless specified throughput values are obtained once a second, making this data collection to run four seconds

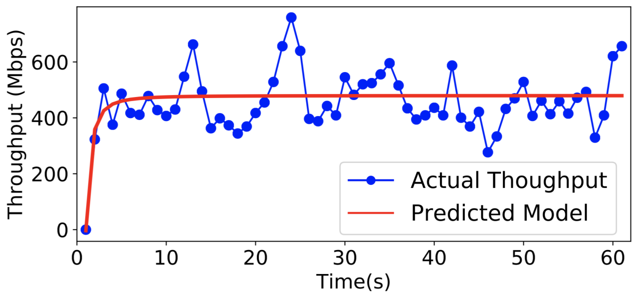


Figure 2: Negative polynomial function.

$$X_t = c + {}_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t. \tag{2}$$

3.3 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 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 data. Compared to AR, ARMA is more robust to recent fluctuations in data as a result of capturing lagged error rate. The ARMA model can capture throughput fluctuations as a result of congestion 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.

$$X_t = c + \varepsilon_t + {}_{i=1}^{p} \varphi_i X_{t-i} + {}_{i=1}^{q} \theta_i \varepsilon_{t-i}. \tag{3}$$

3.4 Autoregressive Integrated Moving Average (ARIMA) Model

Autoregressive Integrated Moving Average (ARIMA) model projects the future values of a series based entirely on its own

Specs	Storage	CPU	Memory (GB)	Bandwidth (Gbps)	RTT (ms)	Transfer Count
XSEDE (Stampede2-Comet)	Lustre	28 x Intel Xeon Gold 6132 @ 2.60 GHz 24 x Intel Xeon E5-1660 @ 3.20 GHz	96 64	10	40	28,209
ESnet	RAID-0	12 x Intel Xeon E5-2643 @3.40GHZ	128	100	89	5,218
Pronghorn	GPFS	16 x Intel Xeon E5-2683 @2.10GHz	192	10	0.1	2,316
HPCLab	NVMe SSD	16 x Intel Xeon E5-2623 @2.60GHz	64	40	0.1	16,383

Table 1: System specification of experiment networks.

inertia. AR part of ARIMA is similar to 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 dataset.

3.5 Adaptive Sampling

Adaptive approach relies on the assumption that throughput will stabilize upon convergence [5]. Thus, it compares consecutive throughput values to determine if a given transfer has reached to convergence point. It stops when the last observed throughput is x% closer to throughput of previous time step. For example, if throughput of a transfer returned 10,20,30,35,45 values in the first five seconds and x is set to = 8%, then adaptive sampling will stop after observing the value of 35 as the ratio between 30 and 35 is less than 8%. Adaptive sampling achieves good performance if the transfer throughput exhibits small 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 at the beginning of the transfer. It will also perform poorly when throughput does not stabilize significantly after reaching to the maximum which can happen when transferring many files since throughput may experience sharp changes when finishing one file and starting to another one due to disk I/O overhead.

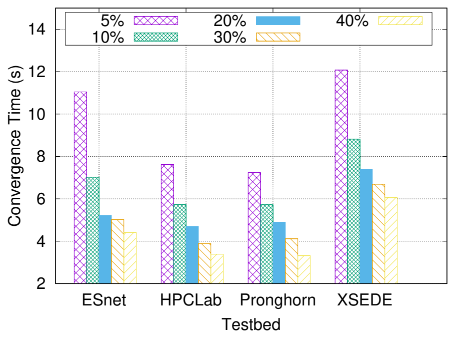
3.6 Fixed Data Size Sampling

Yildirim et al. proposed a model to determine the size of dataset to use in the sample transfers [25]. Unlike the models that we discussed thus far which rely on throughput results to e captured in the runtime to stop sample transfers, fixed-data size solution transfers a small portion of a dataset to run sample transfers. Once the transfer is completed, it measures time which is then used to calculate sampling throughput. To determine the optimal size for sample transfers, Yildirim et al. collected 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 estimates data size to be between 10% and 23% of original dataset size. While this is a promising step to avoid fluctuation in real-time 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 bulk 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.

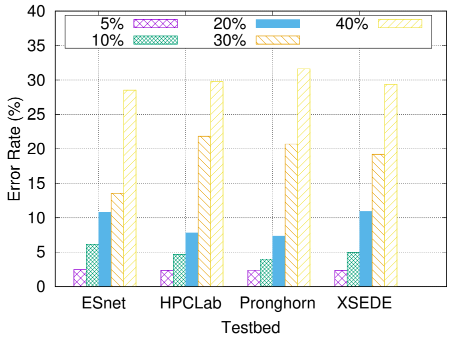
4 DATA COLLECTION

To evaluate the described models in realistic traffic scenarios, we ran file transfers in various local and wide area networks as shown in Table 1 and collected throughput reports periodically. The models are then given a portion of these reports to evaluate their accuracy in predicting actual transfer throughput. 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 parameters, 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 [5, 14, 26, 27]. 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 once a second and saved the reported throughput values of each transfer to a separate file. We used FTP and GridFTP transfers to collect transfer logs in HPCLab, Pronghorn, and ESnet. While GridFTP only supports throughput to be reported at most once a second, our custom FTP protocol allowed us to measure transfer throughput more frequently. Thus, while a majority of experiments in this paper relies on transfer throughput collected in every one second, we evaluated sub-second data collection frequency in Section 5.1.

After running transfers and collecting 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 real-time throughput values, the accuracy of other models is measured as follows. A model is given first three data points from throughput log file to train them. Once the model is trained, it predicts the next data point, which is compared against fourth data point from throughput 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

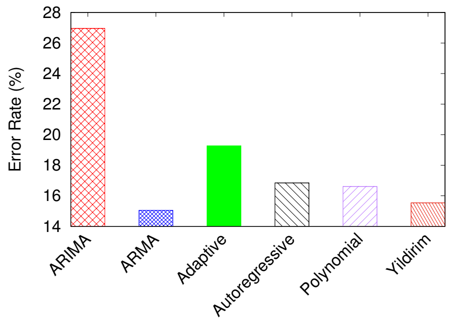


(a) Convergence Time

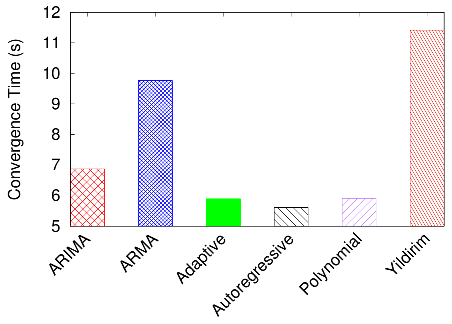


(b) Error Rate

Figure 3: Evaluation of Optimal solution in terms of time and accuracy for various stopping conditions.



(a) Error Rate



(b) Convergence Time

Figure 4: Convergence time and error rate comparison of algorithms for all network results when 10% stopping threshold is used except for fixed data size approach [25]

is retrained with four data points and its convergence is evaluated based on its estimation of fifth data point. The model is trained with more throughput data points until their next data point estimation is close enough to actual data point for the corresponding time. Once the model is able to make accurate estimation of the next data point, the model predicts 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.

4.1 Optimal Solution

As shown in Figure 1, throughput behavior of transfers in different networks differ in terms of convergence time and stability. 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 previously executed 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 3. It is clear that as the threshold increases, it takes longer to convergen 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 it to be close to 10%. However, this is due to the fact that threshold defines the upper bound for the error rate, letting actual error rates to vary 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.

5 EVALUATIONS

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 the algorithms as follows: Adaptive sampling model stops when last two consecutive throughput values are close than a certain percentage, thus the threshold defines 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 which is compared against the actual data point from throughput data. It stops when estimation and actual values are 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 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, thus its error rate and accuracy values are fixed across different threshold values. Finally, we also added result from optimal solution as given in Figure 3 to understand how close each algorithm to the optimal solution.

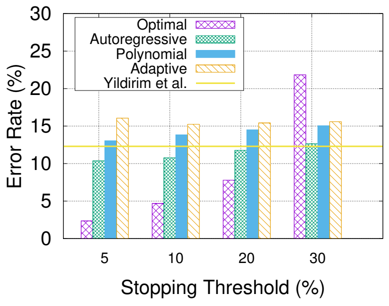
Figure 4 shows the performance comparison of the models when evaluated against all transfer logs collected in all networks. It is clear that ARIMA and Yildirim's model causes up-to 40% higher error rate without no significant improvement in convergence time. While ARMA can obtain around 10% lower error 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 both time and error rate considered considered at the same time. Hence, in the following results, we omit ARMA and ARIMA and focus on the performance evaluations of Autoregressive model.

Figure-5 shows error rate and convergence time of the models in HPCLab testbed. Autoregressive model yields the lowest error rate for all threshold values. As the stopping condition increases from 5% to 30%, the error rate for Autoregressive model 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 earlier. These results indicate that, throughput of transfers in HPCLab experiments converges quickly and exhibit fair amount of fluctuations upon the convergence. Consequently, 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 with 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 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 5(c) shows cumulative distribution function for convergence time with 20% convergence rate where the slow convergence behaviour of Yildirim's model can be observed.

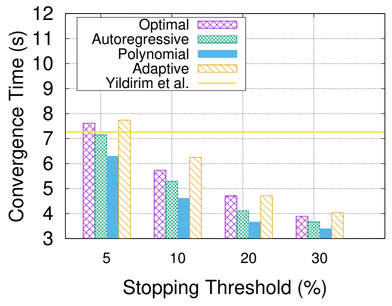
Figure 6 shows the evaluation results for Pronghorn testbed. Similar to HPCLab network, Pronghorn experiments are conducted between two servers in same local area network. While Yildirim's model has the lowest error rate with 6%, its convergence time is nearly 4x higher than other models. Again, Autoregressive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

Figure 7 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 7(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 other 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.

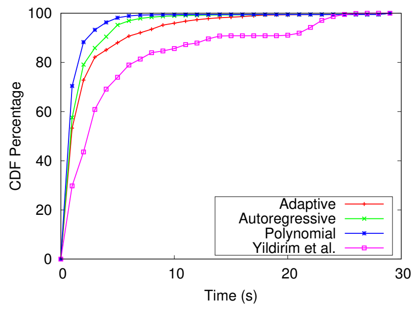
As opposed to other testbeds, XSEDE causes significantly higher error rates due to its shared nature of end system and network resources as shown in Figure 8. While error rates for HPCLab, Pronghorn, and ESnet for Autoregressive were below 15% for all threshold levels, it exceeds to 30% in 30% threshold 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



(a) Error Rate



(b) Convergence Time



(c) CDF Time

Figure 5: Performance comparison of algorithms in HPCLab network transfers. Autoregressive model keeps the error rate below 12% and convergence time below 7 seconds.

Yildirim's model achieves low error rate in all networks, its convergence time is prohibitively high, making it infeasible to use 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.

5.1 Throughput Calculation Frequency

In this section, we investigate the impact of collecting throughput results in sub-second intervals in 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 9(c). Yildirim's model is not shown in the average convergence time and CDF of convergence time figures (Figure 9(b) and 9(c)) as it takes nearly 10x more time than other models as it does not benefit from real-time throughput values and transfers large portion of dataset to run sample transfers.

Figure 9(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 5 and 6, 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.

6 CONCLUSION AND FUTURE WORK

In this work, we propose and evaluate time-series models to estimate throughput of sample transfers and alleviate sampling overhead for real-time transfer optimization. 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 calculating transfer throughput more often than once a second can help to lower convergence time over 6 times while deteriorating error rate by 5-10%. As a potential direction in the future work, we aim to investigate the models in controlled congestion environment to find out how they are affected as background traffic intensifies.

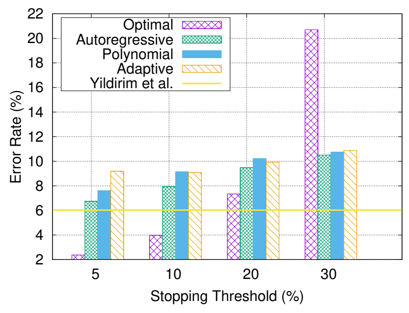
ACKNOWLEDGEMENT

The work in this study was supported in part by the NSF grant OAC-1850353.

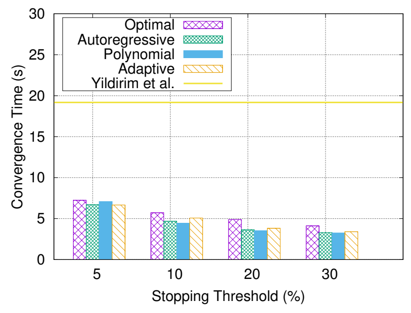
REFERENCES

[1] Ismail Alan, Engin Arslan, and Tevfik Kosar. 2015. Energy-aware data transfer algorithms. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 44.

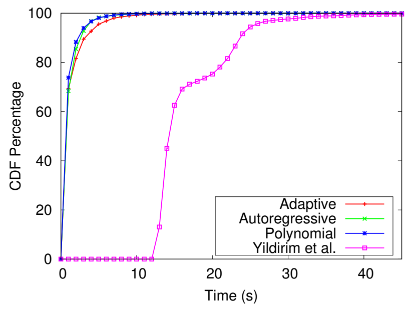
[2] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. 2005. The Globus striped GridFTP framework and server. In *Proceedings*



(a) Error Rate

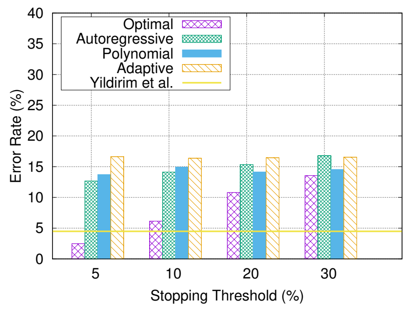


(b) Convergence Time

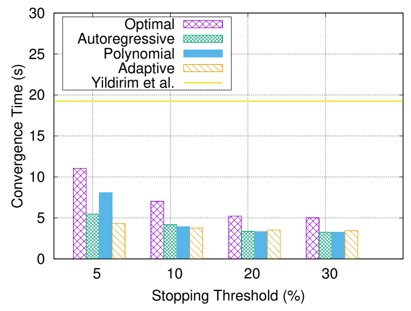


(c) CDF Stopping Time

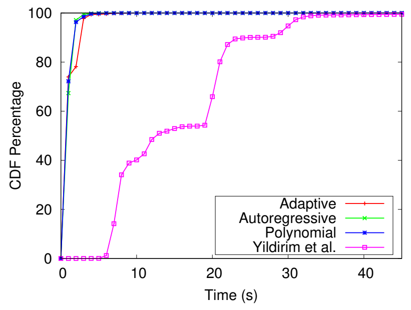
Figure 6: Performance comparison of algorithms in Pronghorn campus cluster. Autoregressive model yields the lower error rate compared to Adaptive and Polynomial models with less than 10% in all thresholds values.



(a) Error Rate

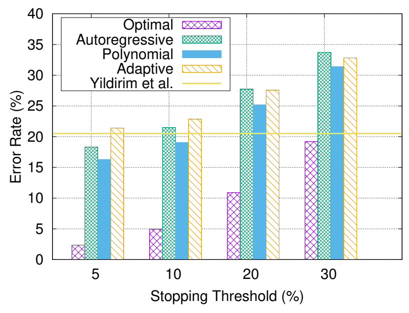


(b) Stopping Time

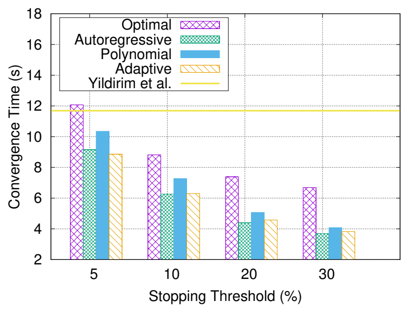


(c) CDF Convergence Time

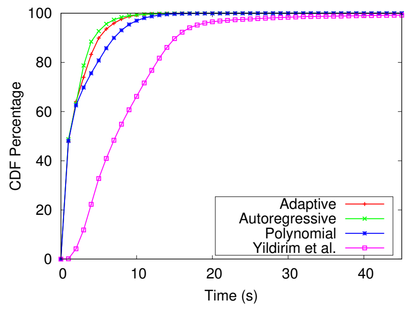
Figure 7: Performance comparison of algorithms in ESnet network transfers. While polynomial model performs similar to Autoregressive model is most cases, its convergence time for 5% threshold is 25% worse.



(a) Error Rate

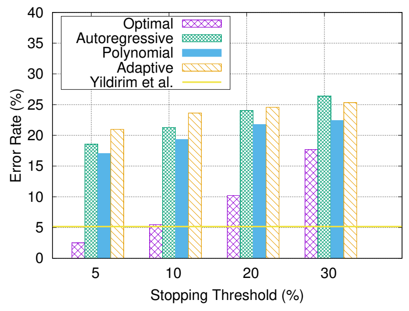


(b) Convergence Time

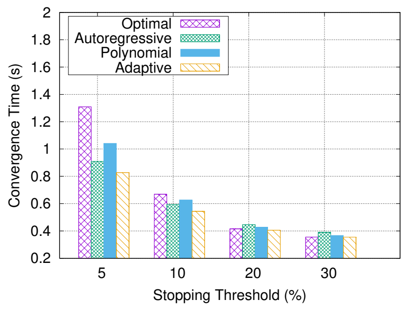


(c) CDF Stopping Time

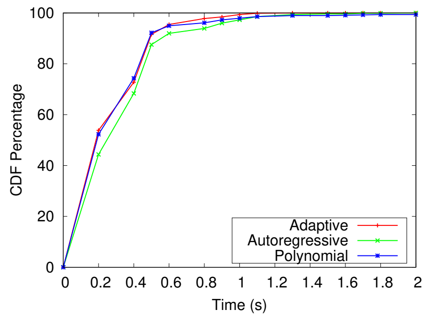
Figure 8: Performance comparison of algorithms in XSEDE network. As opposed to other testbeds, XSEDE causes significantly higher error rates due to its shared nature of end system and network resources.



(a) Error Rate



(b) Stopping Time



(c) CDF Stopping Time

Figure 9: Collecting throughput reports at higher granularity can help to significantly reduce sample transfer time.

of the 2005 ACM/IEEE conference on Supercomputing. IEEE Computer Society, 54.

[3] B. Allen, J. Bresnahan, L. Childers, I. Foster, G. Kandaswamy, R. Kettimuthu, J. Kordas, M. Link, S. Martin, K. Pickett, and S. Tuecke. 2012. Software as a Service for Data Scientists. *Commun. ACM* 55:2 (2012), 81–88.

[4] Engin Arslan, Kemal Guner, and Tevfik Kosar. 2016. 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 (SC '16)*. IEEE Press, Piscataway, NJ, USA, Article 25, 12 pages. http://dl.acm.org/citation.cfm?id=3014904.3014938

[5] Engin Arslan and Tevfik Kosar. 2018. High-Speed Transfer Optimization Based on Historical Analysis and Real-Time Tuning. *IEEE Transactions on Parallel and Distributed Systems* 29, 6 (2018), 1303–1316.

[6] Engin Arslan, Bahadir A Pehlivan, and Tevfik Kosar. 2018. Big data transfer optimization through adaptive parameter tuning. *J. Parallel and Distrib. Comput.* 120 (2018), 89–100.

[7] Engin Arslan, Brandon Ross, and Tevfik Kosar. 2013. Dynamic Protocol Tuning Algorithms for High Performance Data Transfers. In *Proceedings of the 19th International Conference on Parallel Processing (Euro-Par'13)*. Springer-Verlag, Berlin, Heidelberg, 725–736.

[8] Prasanna Balaprakash, Vitali Morozov, Rajkumar Kettimuthu, Kalyan Kumaran, and Ian Foster. 2016. Improving data transfer throughput with direct search optimization. In *Parallel Processing (ICPP), 2016 45th International Conference on*. IEEE, 248–257.

[9] CMS. [n. d.]. The US Compact Muon Solenoid Project. http://uscms.fnal.gov/. ([n. d.]).

[10] N. Freed. [n. d.]. SMTP service extension for command pipelining. http://tools.ietf.org/html/rfc2920. ([n. d.]).

[11] T. J. Hacker, B. D. Noble, and B. D. Atley. 2005. Adaptive Data Block Scheduling for Parallel Streams. In *Proceedings of HPDC '05*. ACM/IEEE, 265–275.

[12] Andreas Hanemann, Jeff W Boote, Eric L Boyd, Jérôme Durand, Loukik Kudarimoti, Roman Łapacz, D Martin Swany, Szymon Trocha, and Jason Zurawski. 2005. Perfsonar: A service oriented architecture for multi-domain network monitoring. In *International conference on service-oriented computing*. Springer, 241–254.

[13] T. Ito, H. Ohsaki, and M. Imase. 2008. On parameter tuning of data transfer protocol GridFTP for Wide-Area Networks. *International Journal of Computer Science and Engineering* 2(4) (Sept. 2008), 177–183.

[14] Rajkumar Kettimuthu, Gayane Vardoyan, Gagan Agrawal, and P Sadayappan. 2014. Modeling and optimizing large-scale wide-area data transfers. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 196–205.

[15] J.T. Kiehl, J. J. Hack, G. B. Bonan, B. A. Boville, D. L. Williamson, and P. J. Rasch. 1998. The National Center for Atmospheric Research Community Climate Model: CCM3. *Journal of Climate* 11:6 (1998), 1131–1149.

[16] Youngjae Kim, Scott Atchley, Geoffroy R Vallée, and Galen M Shipman. 2015. {LADS}: Optimizing Data Transfers Using Layout-Aware Data Scheduling. In *13th {USENIX} Conference on File and Storage Technologies ({FAST} 15)*. 67–80.

[17] Richard J. T. Klein, Robert J. Nicholls, and Frank Thomalla. 2003. Resilience to natural hazards: How useful is this concept? *Global Environmental Change Part B: Environmental Hazards* 5, 1-2 (2003), 35 – 45. https://doi.org/DOI:10.1016/j.hazards.2004.02.001

[18] T. Kosar and M. Balman. 2009. A New Paradigm: Data-Aware Scheduling in Grid Computing. *Future Generation Computing Systems* 25, 4 (2009), 406–413.

[19] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj. 2011. Rate adaptation for adaptive HTTP streaming. In *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 169–174.

[20] Zhengchun Liu, Rajkumar Kettimuthu, Ian Foster, and Peter H Beckman. 2018. Toward a smart data transfer node. *Future Generation Computer Systems* (2018).

[21] MD S. Q. Zulkar Nine, Kemal Guner, and Tevfik Kosar. 2015. Hysteresis-based Optimization of Data Transfer Throughput. In *Proceedings of the Fifth International Workshop on Network-Aware Data Management (NDM '15)*. ACM, New York, NY, USA, Article 5, 9 pages. https://doi.org/10.1145/2832099.2832104

[22] Suraj Pandey and Rajkumar Buyya. 2012. Scheduling workflow applications based on multi-source parallel data retrieval in distributed computing networks. *Comput. J.* 55, 11 (2012), 1288–1308.

[23] Nageswara SV Rao, Qiang Liu, Satyabrata Sen, Greg Hinkel, Neena Imam, Ian Foster, Rajkumar Kettimuthu, Bradley W Settlemyer, Chase Q Wu, and Daqing Yun. 2016. Experimental analysis of file transfer rates over wide-area dedicated connections. In *IEEE 18th High Performance Computing and Communications*. IEEE, 198–205.

[24] E. Yildirim, E. Arslan, J. Kim, and T. Kosar. 2015. Application-Level Optimization of Big Data Transfers Through Pipelining, Parallelism and Concurrency. *Cloud Computing, IEEE Transactions on* PP, 99 (2015), 1–1.

[25] Esma Yildirim, Jangyoung Kim, and Tevfik Kosar. 2013. Modeling throughput sampling size for a cloud-hosted data scheduling and optimization service. *Future Generation Computer Systems* 29, 7 (2013), 1795–1807.

[26] Esma Yildirim and Tevfik Kosar. 2011. Network-aware end-to-end data throughput optimization. In *Proceedings of the first international workshop on Network-aware data management (NDM '11)*. ACM, New York, NY, USA, 21–30. https://doi.org/10.1145/2110217.2110221

[27] E. Yildirim, D. Yin, and T. Kosar. 2011. Prediction of Optimal Parallelism Level in Wide Area Data Transfers. *IEEE TPDS* 22(12) (2011).

[28] Daqing Yun, Chase Q Wu, Nageswara SV Rao, Qiang Liu, Rajkumar Kettimuthu, and Eun-Sung Jung. 2017. Data Transfer Advisor with Transport Profiling Optimization. In *Local Computer Networks (LCN), 2017 IEEE 42nd Conference on*. IEEE, 269–277.