Sample Transfer Optimization with Adaptive Deep Neural Network

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

Md Arifuzzaman
Computer Science and Engineering
University of Nevada Reno
Nevada, USA
arif@nevada.unr.edu

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

***Abstract*—Application-layer transfer configurations play a crucial role in achieving desirable performance in high-speed networks. However, finding the optimal configuration for a given transfer task is a difficult problem as it depends on various factors including dataset characteristics, network settings, and background traffic. The state-of-the-art transfer tuning solutions rely on real-time sample transfers to evaluate various configurations and estimate the optimal one. However, existing approaches to run sample transfers incur high delay and measurement errors, thus significantly limit the efficiency of the transfer tuning algorithms. In this paper, we introduce adaptive feed forward deep neural network (DNN) to minimize the error rate of sample transfers without increasing their execution time. We ran 115K file transfers in four different high-speed networks and used their logs to train an adaptive DNN that can quickly and accurately predict the throughput of sample transfers by analyzing instantaneous throughput values. The results gathered in various networks with rich set of transfer configurations indicate that the proposed model reduces error rate by up to 50% compared to the state-of-the-art solutions while keeping the execution time low. We also show that one can further reduce delay or error rate by tuning hyperparameters of the model to meet specific needs of user or application. Finally, transfer learning analysis reveals that the model developed in one network would yield accurate results in other networks with similar transfer convergence characteristics, alleviating the needs to run an extensive data collection and model derivation efforts for each network.**

***Index Terms*—Sample transfer, Network probing, Transfer modeling, Deep Neural Network**

I. Introduction

Driven by advancements in instrument technologies and computing power, many scientific applications have started to generate massive volumes of data [1]–[4]. For example, high energy physics and particle experiment ATLAS produces 1 PB of data in each second, which is reduced to 1-2 GB after filtering [5]. The Dark Energy Survey instrument captures the pictures of the southern sky and generates nearly 500 GB of data each night [3]. The successor of the Dark Energy Survey project, Large Synoptic Survey Telescope [4], will use 3.2 gigapixel cameras and produce 30 TB data every night. Enormous amount of data collected by these science projects often needs to be streamed to remote computing and storage facilities to be processed and archived. Although high-performance networks with up to 100 Gbps capacity has been established to keep up with increasing data transfer rates, it has been found challenging to attain full network utilization in these networks [6], [7].

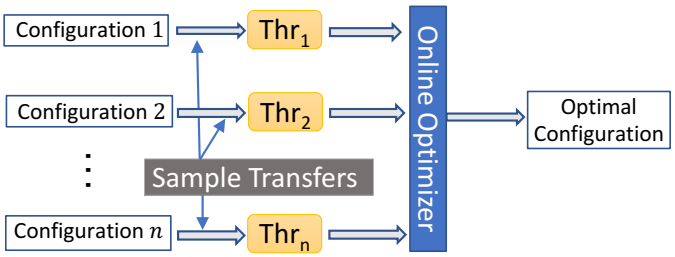


Fig. 1. Online optimization algorithms rely on sample transfer to evaluate various application layer transfer configurations to discover the ideal one.

Researchers have proposed online optimization algorithms to tune transfer configurations in real-time at the transport and application layers without relying on historical data [7], [8]. They work by evaluating various transfer configurations using sample transfers and cost functions. Then, the output of sample transfers (e.g., throughput and loss rate) are used to predict an optimal configuration that would satisfy user's expectations such as high throughput and low delay as illustrated in Figure 1. While running sample transfers to evaluate transport layer configurations is relatively easy due to being able to access to a large set of network metrics (e.g., RTT, throughput, loss rate etc.) at fine granularity (i.e. in the orders of milliseconds), it is rather challenging for application layer configurations since many commonly used transfer applications (e.g., scp, FTP, and GridFTP) report only few metrics (e.g., transfer completion) at coarse granularity (i.e., in the orders of seconds). Despite these challenges, previous studies have shown that one can achieve more than an order of magnitude improvement in transfer throughput by tuning application layer parameters where transport-layer solutions are either not available or insufficient.

A typical approach to conduct sample transfers is to transfer one or more files using the configuration that is being tested (aka *fixed-size*) [9]. However, determining the amount of data (i.e. the number of files and file size) that is sufficient enough to evaluate the performance of a configuration requires significant upfront work since optimal data size depends on transfer conditions such as network bandwidth and storage

I/O performance. Even one-time data collection and model derivation may not work well as such models are inherently susceptible to modifications in network and end systems such as bandwidth increase, network path change, and file system upgrade as they need to be re-calibrated with new data to adapt the changes. Yet, experimental results indicate that fixed-size sample transfers causes prohibitively long execution time even after data size optimizations, thus, they do not offer a feasible solution for online optimization algorithms in today's fast evolving production networks.

To avoid performance and training costs of fixed-size sample transfer models, fixed-duration [10], adaptive [11], and time-series [12] models have been proposed. Fixed-duration approach evaluates different configurations based on the amount of data transferred within a fixed time period (e.g., five seconds). Similar to fixed-size approach, fixed-duration ,method requires a fine tuning of time duration that sample transfers will run since short time periods will result in poor accuracy while long ones will lead to extended execution times. Adaptive sampling method starts sample transfer and periodically (e.g., once a second) monitors transfer throughput to identify convergence time. The idea behind this is that throughput of a transfer is supposed to reach to a maximum speed and stay within a reasonable range afterwards. By keeping track of reported throughput values in sliding windows, the adaptive model aims to detect when throughput values in a window follows a stable pattern at which point, it can stop the transfer and use the average throughput of the last window as the throughput of a given configuration. Although the adaptive approach promises fast convergence time with low measurement error in dedicated networks, it fails to make a timely decision in shared networks where transfer throughput exhibits fluctuating behavior due to congestion and inaccurate throughput measurements. Finally, in a previous work, we modeled sample transfers with Autoregressive (AR), Autoregressive Moving Average (ARMA), Autoregressive Integrated Moving Average (ARIMA) time-series analyses to capture behaviour of instantaneous throughput reports and forecast future values. Among AR, ARMA, and ARIMA, AR obtained the best performance by reducing the sample transfer time below six seconds, however, its error rate can go up to 30%, potentially causing online optimization algorithms to make inaccurate conclusions.

In this paper, we propose an adaptive deep neural network (DNN) to minimize execution time of sample transfers while keeping error rate low. DNN offers a unique capability to extract complex patterns to accurately predict convergence time and throughput of sample transfers in wide range of conditions where analytical and heuristic models are unable to perform well. We trained an adaptive DNN model to take instantaneous throughput metrics from sample transfer as inputs to predict convergence time and throughput as outputs. Evaluations in rich network and dataset settings reveal that the adaptive DNN is able to improve accuracy of sample transfers by up to 50% compared to the state-of-the-art solutions in exchange of slight increase in execution time.

The rest of paper is organized as follows: Section II explains the motivation and presents related work in the area of sample transfer optimization. Section III describes the data collection process and Section IV details the proposed DNN model. Section V discusses the evaluation results. Finally, Section VI concludes the paper with the summary.

II. Related Work

Throughput of file transfers depends on many factors including but not limited to dataset characteristics (e.g., file count and average file size), network settings (e.g., bandwidth and round trip time), transfer configurations (e.g., number of parallel streams and concurrent file transfers), and background traffic [11], [13]–[15]. Therefore, it is difficult to model and predict the throughput of a transfer ahead of time. Sample transfers help to estimate throughput of long transfers by running *micro* transfers for a short period of time, however fine tuning the scale of sample transfers (i.e., duration, data size, etc.) is rather challenging due to conflicting nature of execution time and accuracy. As the duration of sample transfers is increased, the accuracy of throughput estimation tend to increase as well. Although running sample transfers for an extended period of time improves the accuracy of performance estimation, it causes several issues: First, if sample transfers are used to check the health of a production network (as done by perfSonar [16]), then prolonged execution time of sample transfers could negatively affect the production workload. Second, the performance gain offered by real-time transfer optimization algorithms that rely on sample transfers to identify an optimal transfer configuration starts to disappear as the cost of sample transfers surge since the optimization process involves assessing the performance of suboptimal configurations. Previous studies proposed fixed-size [9], [16], fixed-duration [10], [17], adaptive [11], [18], and time-series analysis [12] methods to execute sample transfers.

PerfSonar is developed to execute periodic measurements mainly between research and education institutions that are connected with high-speed networks [16]. It collects throughput, delay, packet re-transmission, and packet route information using measurement tools such as *traceroute* and *Iperf*. Its end-to-end throughput measurement is done by transferring a single file (by default 10GB in size) transfer between end points once in every six hours. However, the choice of using fixed-size file for sample transfer causes it to underestimate achievable throughput since it is unable to run long enough to reach maximum speed when the bandwidth is higher than 2-3 Gbps. Moreover, using one file in sample transfers prevents sample transfers such from discovering potential throughput improvements by means of concurrent file transfers, which helps to improve I/O performance in parallel file systems.

Yildirim et al. proposed a regression analysis to model dataset size for fixed-size sample transfer approach [9]. They first run extensive experiments to collect accuracy statistics for various dataset size values, then run linear regression to derive a relationship between sample transfer size and transfer settings such as bandwidth, RTT, and average file size. As

TABLE I
SYSTEM SPECIFICATION OF EXPERIMENT NETWORKS.

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	64	10	40	53,796
ESnet	RAID-0	12 x Intel Xeon E5-2643 @3.40GHZ	128	100	89	16,849
Pronghorn	GPFS	16 x Intel Xeon E5-2683 @2.10GHz	192	10	0.1	3,000
HPCLab	NVMe SSD	16 x Intel Xeon E5-2623 @2.60GHz	64	40	0.1	41,768
Total						**115,413**

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 original dataset size which would incur too much delay for bulk transfers and result in inaccurate sampling size for small ones. Moreover, real-time transfer optimization algorithms would end up transferring the most, if not all, of whole dataset during search phase in case of requiring multiple sample transfers, making the discovery of optimal transfer configuration useless.

Alan et al. introduced the fixed-duration method to measure throughput of sample transfers to determine the configuration with the highest throughput to energy consumption ratio [10]. The authors claim that five 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 it can take up to 20 seconds for some transfers to converge due to connection setup and slow-start costs especially in long fat networks. Balaprakash et al. proposed a direct search algorithm to tune transfer parameters in real-time [17]. They use control epochs to run sample transfers and measure corresponding throughput for different configurations. To increase accuracy, they set the control epoch time to 30 seconds which severely limits the applicability of real-time optimization algorithms for most transfers in scientific networks as they are dominantly short-lived [19].

In a previous work, we proposed an adaptive sampling technique which starts transferring an entire dataset and monitor throughput periodically [11]. 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 transfer throughput does not exhibit too much fluctuations upon convergence. However, our experiments proved that this assumption does not hold true in shared networks where I/O and network resource contention is significant and unpredictable. In another study, we conducted time-series analysis for instantaneous throughput results to model and predict future transfer throughput. The experimental results indicate that among other time-series models, Autoregressive is able to predict the convergence time in less than six seconds, however its error rate reaches to more than 20% in shared, production networks.

III. DATA COLLECTION

To train the adaptive DNN model, we ran 115K file transfers in four networks whose specifications are given in Table I. HPCLab servers and Pronghorn campus cluster are located at the University of Nevada, Reno (UNR). HPCLab servers are equipped with direct attached NVMe SSDs whereas Pronghorn servers are backbed by GPFS distributed file system. ESnet and XSEDE transfers represent wide-area network conditions with 89 ms and 40 ms delay between end points. We used datasets that consists of files with various sizes (ranges between 1 MB and 100 GB) and counts (ranges between 1 and 10,000) to capture the impact file size and count in sample transfers. We also tuned application layer parameters such as concurrency and parallelism to assess their impact. Parallelism defines the number of network connections for single file transfer whereas concurrency sets the number of concurrent file transfers. XSEDE and Pronghorn networks due to which they are exposed to storage, server, and network interference. On the other hand, ESnet and HPCLab networks are isolated environments hence less exposed to unpredictable resource interference.

We used GridFTP in XSEDE network and custom file transfer application in others and collected instantaneous throughput logs as reported by the transfer application. Since GridFTP sends throughput updates at most once a second, XSEDE logs contain throughput reports on second-basis whereas HPCLab, ESNet, and Pronghorn logs include instantaneous throughput values both second and sub-second (i.e. 100ms) granularity with the help of custom transfer applications. We evaluated the impact of obtaining instantaneous throughput reports in sub-second basis in Section V-A, therefore the rest of evaluations are derived from the transfer logs with second-basis throughput reports. Due to shared nature of XSEDE and Pronghorn networks, some transfers contained significantly fluctuating reports such as 1000 Mbps in one second and 0 Mbps in the following. To detect and remove such outliers, we excluded transfers whose standard deviation is more than 50% of average throughput. This process removed nearly 32K transfers and remaining 83K transfer logs are split as training (70%) and test (30%) data to train and test the proposed DNN model.

Defining the Optimal Solution

Throughput of sample transfers exhibit distinct patterns in terms of convergence time and stability due to variation in network and end system conditions such as constant background traffic in XSEDE network. Thus, we define the *optimal*

solution that can be used to better evaluate the performance different models since throughput reports are inherently more unstable in some networks, hampering accurate estimations in short time frame. We first calculate the average throughput of a whole transfer by dividing the data size to transfer time. Then, the *optimal* solution scans per-second basis throughput reports starting from first second to identify the time at which throughput is close to actual average throughput. Once found, the time is assumed to be the minimum time a sample transfer can take to accurately estimate the throughput for given dataset transfer. The accuracy of the *optimal* solution is calculated as the percentage of difference between the throughput of optimal time and average throughput of whole transfer.

We evaluated various threshold values to measure closeness of an instantaneous throughput value to actual average throughput and found that lower threshold leads to higher execution time in exchange of lower error rates, as expected. While 5% threshold leads to 12 seconds execution time, 40% threshold reduces execution time to less than 5 seconds in return of increased error rate. Since, we want to find a model that can estimate sample transfer throughput within a reasonable time frame and error rate, we used 10% threshold to compare against the models in the next section.

IV. Adaptive Deep Neural Network

We aim to reduce the execution time of sample transfer by analyzing instantaneous throughput reports that are populated in every second. Hence, the DNN model will be given real-time throughput readings to predict the throughput of a sample transfer upon convergence. Since sample transfers converge at different times in different networks, the model needs to determine if a sample transfer has converged or not in real-time, which requires an ability to make predictions using a various number of instantaneous throughput reports. For example, at the end of first second of a sample transfer, the model is expected to use single throughput report to forecast convergence throughput of the given transfer. However, if its prediction confidence is low due to lack of information, then we let the transfer continue running to gather more data and make new predictions using more throughput reports. We noticed that throughput of 98% of all transfers in historical data converge between 3 and 16 seconds, thus the adaptive model consists of 14 DNNs (from 2-input to 15-input) as shown in Figure 2.

Rather than using absolute throughput values as inputs to the DNN models, we use percentage of change between consecutive throughput values to minimize model bias. For example, average throughput of HPCLab transfers is around 16 Gbps whereas it is less than 5 Gbps for XSEDE transfers. Thus, training a DNN model using absolute throughput values could cause the model to assume a relationship between absolute throughput values and convergence time and potentially hampers its use in other networks. Instead, if the model can learn throughput convergence behaviour by analyzing rate changes, it can be transferred to other networks with similar convergence behaviour even though network bandwidth is

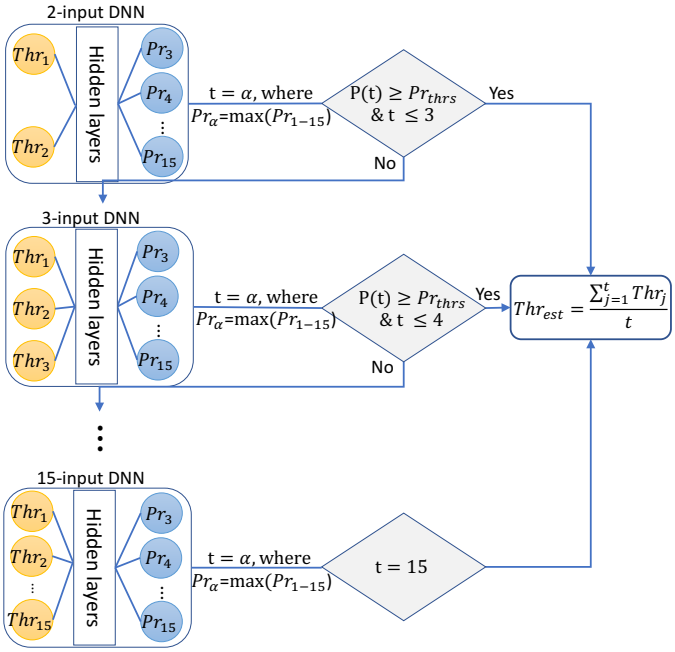
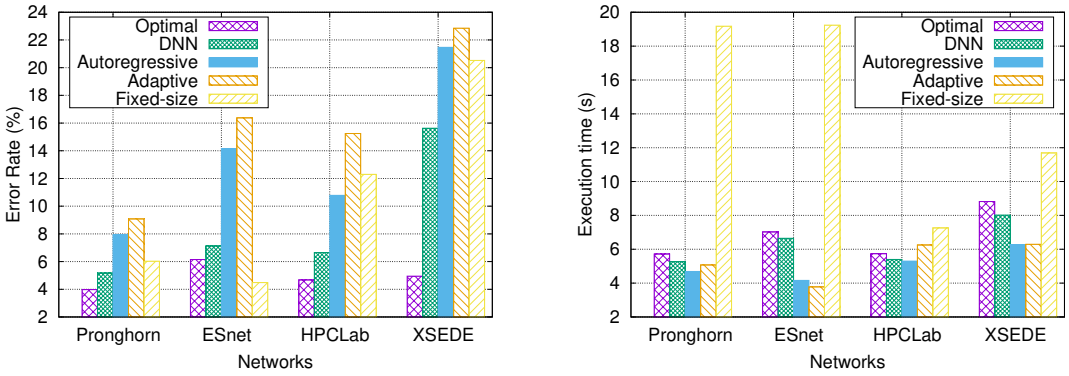


Fig. 2. The Adaptive Deep Neural Network starts estimating the probability of convergence times (P_i) by using throughput values from first three seconds. If the maximum estimated probability (P_α) is greater than a specified threshold (Pr_{thrs}) and convergence time t is smaller than current time, then the adaptive DNN assumes convergence is satisfied and takes average throughput of t seconds as the throughput of a sample transfer.

different. To give an example on rate change calculations, consider a sample transfer with $\{200, 400, 500, 450\}$ Mbps throughput reports at its fourth second. We convert these values to rate changes using Equation 1, which returns $\{100\%, 25\%, -10\%\}$ values. Note that this reduces the number of inputs by one as we can calculate first rate change using two second throughput values. The output of the model is a convergence time classifier to which is then translated into convergence throughput by taking the average of observed throughput results.

$$Rate = \frac{Throughput_{current} - Throughput_{previous}}{Throughput_{previous}} \quad (1)$$

We trained the adaptive model using historical data logs with *optimal* solution as the output. For example, if throughput of first six seconds of a transfer is reported as $\{100, 220, 560, 610, 550, 600\}$ Mbps with actual average throughput of whole transfer is being 605 Mbps, the optimal solution will mark the fourth second as the optimal execution time as it reaches the 10% range of actual throughput in the fourth second. Then, we train a 2-input DNN with first two rate changes, $\{120\%, 154\%\}$, and output as 4, 3-input DNN with first three rate changes, $\{120\%, 154\%, 9\%\}$, and output as 4 and so forth such that the generated DNN models can learn various rate change patterns that leads to 4 second convergence time. The classifier is then able to identify unique patterns for convergence time based on rate change input values.



(a) Error Rate (b) Execution Time

Fig. 3. Performance comparison of algorithms in different networks. The DNN models obtains close-to-optimal results for execution time and error rate in most cases.

To use the model in real-time, we first pass the two rate change values of the sample transfer to the 2-input model at $t = 3s$, which estimates the probabilities of convergence time to be 3-16 seconds. If the probability of 3s is greater than a certain than a threshold (Pr_{thrs} in Figure 2), we terminate the sample transfer immediately and take the average of all three throughput reports as the throughput of the sample transfer. Otherwise, we wait for another second to try 3-input DNN to see if it can be make a prediction with high confidence for output of 3s or 4s. Note that it is possible that a DNN model can estimate the convergence time to be in the future. For example, 3-input DNN can predict convergence time to be 10 seconds. While this allows us to possibly terminate the sample transfer early, we also need to estimate the throughput value at 10s which is not possible with the classifier DNNs. We tried several regression DNN models to predict throughput value of future time points, however, the accuracy of regression models tend to be very low, thus we instead let the transfer continue its execution. If an estimated convergence time is lager than current time, we continue gathering instantaneous transfer throughput in the next time step and testing the corresponding DNN model rather than simply waiting until the estimated convergence time. For instance, even if 3-input DNN estimates convergence time to be 10^{th} second with high confidence, we keep running 4-input DNN at 4^{th} second and 5-input DNN at 5^{th} second to see if those models could potentially estimate convergence time to be less than 10 seconds with high confidence. Once the adaptive DNN model estimates the convergence time to be equal or less than current time (i.e., the number of inputs is greater than or equal to estimated convergence time) with high probability, we then terminate the sample transfer and take the average of all throughput reports up-to current time to estimate the average throughput of the sample transfer.

By default, all neural networks in the proposed model consists of 5 layers including input layer and output layer where all three hidden layers contain 100 intermediate nodes. We use Limited-memory BroydenFletcherGoldfarbShanno (lbfgs) as our solver and Rectifier Linear Unit (RELU) as the activation function with the learning rate of 0.001. We evaluate the performance of the models based on various training configurations in Section V-B.

V. Evaluations

In this section, we evaluate the performance of different models in terms of execution time and estimation accuracy. We compared the adaptive DNN model against following solutions: (i) *Fixed-size* approach that transfers a predetermined size of data to run sample transfers and measures time to calculate their 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 sample transfer size to file size and bandwidth-delay-product. The model, on average, estimates data size to be between 10% and 23% of original dataset size [9]. (ii) *Adaptive sampling* approach which assumes that instantaneous throughput values stabilizes upon convergence [11]. Thus, it compares consecutive throughput values to decide if a given transfer has converged or not. (iii) *Autoregressive* (AR) model that uses throughput observations from previous time steps as input to predict the throughput at the next time step. As opposed to fixed-size approach, AR is trained in the real-time using instantaneous throughput observations. Its execution time is calculated by comparing its throughput prediction for the next time point with actual throughput. If its prediction accuracy is less than a certain threshold, then AR models is assumed to be accurate and its estimation for next 10 seconds are used to calculate the throughput of sample transfer. We used 15% of dataset as data size of fixed-size method and 10% threshold for adaptive sampling and AR models when evaluating their performance.

Figure 3 shows the performance comparison of the models when evaluated in different networks. It is clear that fixed-size approach causes up-to 19 seconds execution time in Pronghorn and ESnet networks whereas its error rate reaches to 20% in XSEDE network. In contrast, adaptive sampling method yields smaller execution times in all networks with less than 6 seconds in the worst case. In exchange, its accuracy degrades significantly and reaches to more than 20% in XSEDE. In

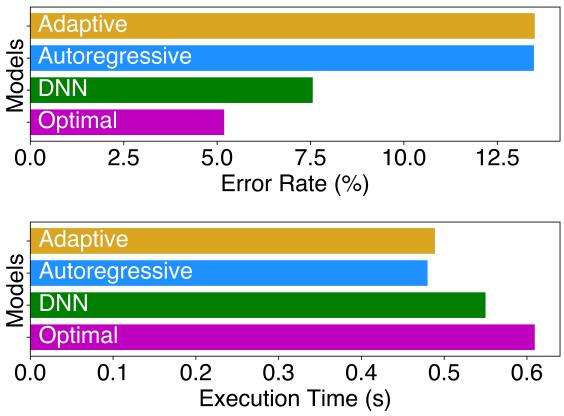


Fig. 4. Comparison of sampling algorithms with sub-second data collection frequency

overall, error rate of adaptive sampling approach is 2-3x higher than the optimal one. AR model yields the lowest execution time for sample transfers in all networks whereas its error rate is also much higher compared to the optimal and the DNN model. Although AR is able to keep the convergence time less than six seconds in all networks, its error rate is more than 20% in XSEDE and considerably higher than optimal value in other networks. For example, optimal error rate in ESnet is around 6% whereas AR causes 14% error.

On the other hand, the DNN model yields the close to *optimal* results in almost all networks. Its error rate is within 1-2% range of *optimal* in Pronghorn, ESnet, and HPCLab networks. However, the error rate jumps to more than 14% in XSEDE network. Although 14% is far from optimal, it is still the lowest error rate among all algorithms. The smallest error rate in XSEDE network after the DNN model is greater than 20% by the fixed-size approach. As opposed to other testbeds, XSEDE causes significantly higher error rates due to its shared nature of end system and network resources. Except XSEDE, the error rate of the DNN model is less 7% which is nearly 50% improvement over AR model whose error rate ranges between 8-21.5%. In overall, the DNN model enhances the error rate prediction over AR by a significant margin whose error rate cannot be enhanced much despite increased threshold values [12].

In terms of execution time, the DNN always yields lower results compared to optimal solution. For example, the optimal solution estimates the convergence time to be 5.9s for Pronghorn whereas it is 5.4s for the DNN. Compared to AR and Adaptive, the DNN has higher execution time values in all networks though the difference is no more than 2 seconds at most. As an example, AR yields 5.6s execution in HPCLab while causing 10.9% error rate whereas the DNN model yields 5.7s execution in exchange of 6.3% error rate, more than 30% improvement over AR.

A. Sub-second Throughput Measurements

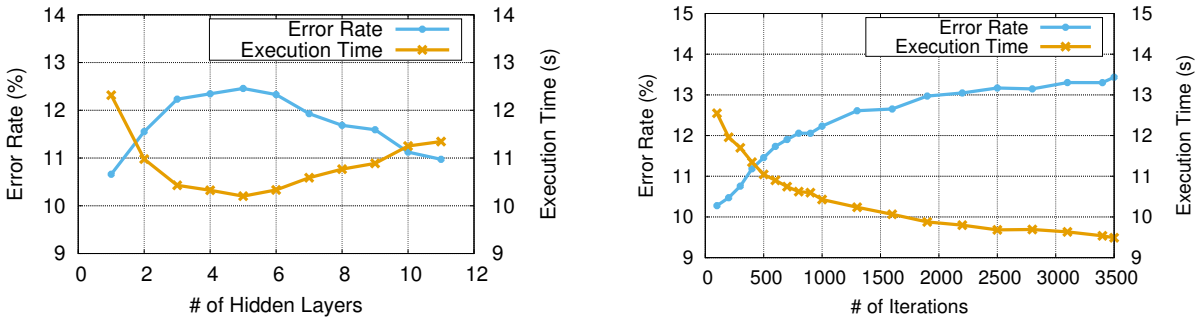
In this section, we investigate the impact of collecting throughput results in sub-second intervals in attempt to achieve faster convergence time which is critical to be able to run many sample transfers in a timely manner. While GridFTP servers do not report instantaneous throughput information in sub-second intervals, we configured our custom transfer application calculate and log transfer throughput in every 100ms in HPCLab, Pronghorn, and ESnet networks. More frequent data collection allows to incorporate larger number of inputs to the models without causing long execution time as shown in Figure 5. While convergence time in any network with one second data collection frequency cannot be lower than four seconds, it falls less than 0.6 seconds when data collection frequency is increased to 100ms, reducing the execution time by more than an order of magnitude. While sub-second throughput values increases error rate of AR and Adaptive sampling methods to more than 10%, the adaptive DNN model manages to keeps its error rate below 8% in exchange of less than 0.1 second difference in execution time.

B. Hyperparameter Tuning

In this section, we investigate the impact of the DNN model parameters on error rate and execution time with the goal of discovering the parameter setting that leads to low execution time and error rate. Among many hyperparameters, we found that the number of hidden layers and the number of maximum iterations have the highest impact on the performance of the model. Figure 5(a) shows that the execution time starts to decrease as more hidden layers are added in exchange of increasing error rate. Execution time is minimized when there are four hidden layers which leads to highest error rate in return. Since we want to choose the number of hidden layers that yields low error rate and low execution time, we used three hidden layers in all DNN models for all networks as more hidden layers do not offer obvious gain when execution time and error rate metrics are considered together. The three hidden layers result in 12.23% error rate and 10.8 seconds execution time on average for all networks experiments. While we can achieve similar performance with 10 hidden layers as well, higher hidden layers increases the cost training time of the model.

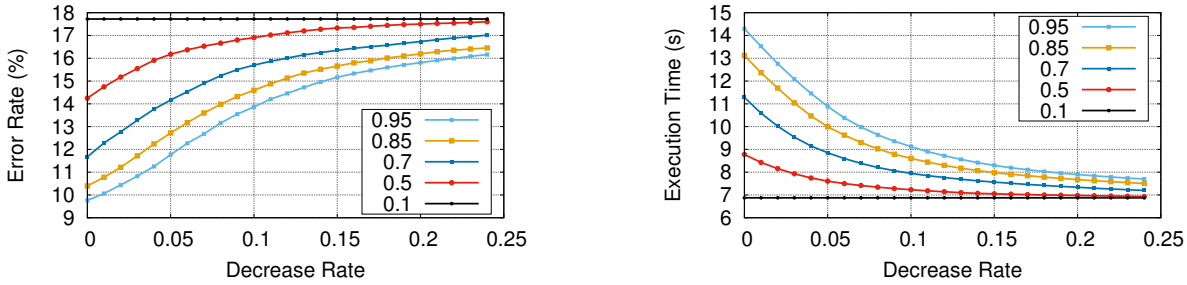
Figure 5(b) illustrate error rate and execution time for different values of maximum iteration. As with the number of hidden layers, execution time starts to decrease when the maximum number of iterations is increased. However, error rate is also increasing when higher values of maximum iterations is used. While maximum iteration of 100 yields 10.2% error rate with 12.7s execution time, these value become 13.4% and 9.5s for 3500 maximum iteration setting. As a result, we set the value of maximum number of iterations to 1000 at which point error rate is 12.1% and execution time is 10.8 seconds for all networks.

As shown in Figure 2, the adaptive DNN model requires the probability of an estimated convergence time value to be greater than a threshold to avoid making poor decisions. As expected, high threshold reduces error rate while increasing execution time as shown in Figure 6. In addition to defining a threshold, we also introduce *Decrease Rate* which aims to adjust the threshold over time. Since it's less likely for



(a) Hidden Layers (b) Maximum Iteration

Fig. 5. The impact of model hyperparameters on error rate and execution of the DNN.



(a) Error Rate (b) Execution Time

Fig. 6. The impact of model classifier threshold on error rate and execution of the DNN in XSEDE network.

a sample transfer to converge in three seconds, we set high threshold value for 2-input DNN and gradually lower as time passes. As an example, 0.95% initial threshold with 0.05% decrease rate will use threshold of 0.95% for 2-input DNN, 0.9% for 3-input DNN, and 30% for 15-input DNN. The decrease rate of 0% refers static threshold scenario where same threshold value is enforced at every input level of the model. We evaluated the impact of initial threshold and decrease rate values in XSEDE network where error rate of the DNN model showed the highest variation compared to the optimal one.

The initial value of 0.1% with 0% decrease rate reduces execution time to less than 7 seconds while causing error rate to be slightly lower than 18%. Compared to the performance of AR in XSEDE, it is 0.5 seconds higher execution time in return of 4% improvement in error rate. Moreover, 0.95% static threshold yields less than 10% error rate while causing execution time to rise above 14 seconds. As a result, one can tune the threshold value to adjust desired level execution time and error rate. For example, an online optimization algorithm can define low threshold to run many sample transfer quickly if less than 20% error rate is acceptable.

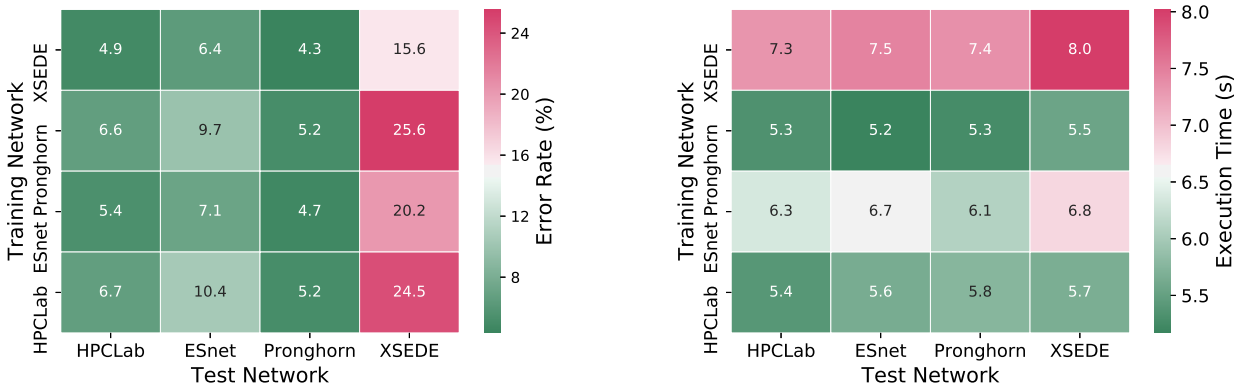
C. Analysis of Transfer Learning

One of the main drawbacks of machine learning models is poor versatility when not configured properly. In an attempt to increase adaptability of the DNN models that are trained in one network condition in other networks, we adopted throughput rate change as inputs to DNN models rather than absolute values as discussed in Section IV such that the models will not be biased toward absolute throughput values.

We evaluated the performance of models that are trained in one network by testing in other networks as demonstrated in Figure 7. The error rate for XSEDE network reaches to 24% when the model is trained with Pronghorn network logs. However, the error rate for the other networks does not degrade more than 1-2% when they are tested with the models that are trained in different networks. On the other hand, execution time reduces when the HPCLab and Pronghorn trained models are used in any other network. This is an expected outcome as HPCLab and Pronghorn transfers converge in less than six seconds as shown in Figure 4. As an example, when XSEDE-trained model is used to test XSEDE transfers, it takes 8 seconds to converge. On the contrary, Pronghorn-trained model yields 5.5 seconds of execution time when tested against XSEDE transfers. In overall, HPCLab and Pronghorn networks exhibit similar behavior and does so XSEDE and ESnet networks. So, when HPCLab-trained model is used in Pronghorn network, the results do not deviate much compared to the performance of HPCLab-trained model. Similar observations can be made between ESnet and XSEDE. Therefore, the proposed model offers promising results for transfer learning as long as the training dataset involves networks with similar characteristics in terms of convergence behavior.

VI. Conclusion and Future Work

In this work, we propose an adaptive deep neural network model to predict throughput of sample transfers with high



(a) Error Rate

(b) Execution Time

Fig. 7. Heat map for the DNN models that are trained and tested in different networks show that the proposed solution supports transfer learning

accuracy in short period of time. The proposed model process instantaneous throughput reports to extract relationship between consecutive throughput values and convergence time. The results in wide range of network, dataset, and configuration settings reveal that the DNN models is able to perform close-to optimal in almost all cases with less than 6.5% error rate and less than 8 seconds sample transfer execution time. Compared to the Autoregressive model, it yields more than 40% improvement with less than 10% increase in execution time. We further evaluated the impact of gathering throughput reports more frequently than current-reporting frequency (i.e. one second) and observed more than orders of magnitude improvement in execution time without degrading accuracy. We also analyzed various hyperparameters and model configurations to show that the DNN model is versatile and able to offer wide range of error rate and execution time trade-off such that users can define their priority to lower error rate or execution time. Moreover, we investigated applicability of the models to new networks and discovered that it is transferable without much of loss in error rate and execution time as long as tested networks shows similarity to trained networks in terms of convergence behavior.

ACKNOWLEDGEMENT

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

REFERENCES

[1] M. Wolf, G. Eisenhauer, and P. Widener, "Rethinking streaming system construction for next-generation collaborative science." Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2016.

[2] J. Loveday, "The sloan digital sky survey," *Contemporary Physics*, vol. 43, no. 6, pp. 437–449, 2002.

[3] "Dark Energy Survey," 2017, https://www.darkenergysurvey.org/.

[4] "Large Synoptic Survey Telescope," 2017, https://www.lsst.org/.

[5] "A Toroidal LHC ApparatuS Project (ATLAS)," http://atlas.web.cern.ch/.

[6] E. Arslan, B. Ross, and T. Kosar, "Dynamic protocol tuning algorithms for high performance data transfers," in *European Conference on Parallel Processing*. Springer, 2013, pp. 725–736.

[7] E. Yildirim, E. Arslan, J. Kim, and T. Kosar, "Application-level optimization of big data transfers through pipelining, parallelism and concurrency," *IEEE Transactions on Cloud Computing*, vol. 4, no. 1, pp. 63–75, 2015.

[8] 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.

[9] 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.

[10] I. Alan, E. Arslan, and T. Kosar, "Energy-aware data transfer algorithms," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 44.

[11] E. Arslan and T. Kosar, "High-speed transfer optimization based on historical analysis and real-time tuning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 6, pp. 1303–1316, 2018.

[12] H. Sapkota, B. A. Pehlivan, and E. Arslan, "Time series analysis for efficient sample transfers," in *Proceedings of the ACM Workshop on Systems and Network Telemetry and Analytics*. ACM, 2019, pp. 11–18.

[13] Y. Liu, Z. Liu, R. Kettimuthu, N. S. Rao, Z. Chen, and I. Foster, "Data transfer between scientific facilities–bottleneck analysis, insights, and optimizations," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2019, pp. 122–131.

[14] 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.

[15] M. S. Z. Nine, K. Guner, Z. Huang, X. Wang, J. Xu, and T. Kosar, "Big data transfer optimization based on offline knowledge discovery and adaptive sampling," in *Big Data (Big Data), 2017 IEEE International Conference on*. IEEE, 2017, pp. 465–472.

[16] 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.

[17] 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.

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

[19] Z. Liu, R. Kettimuthu, I. Foster, and N. S. Rao, "Cross-geography scientific data transferring trends and behavior," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2018, pp. 267–278.