High-Speed Transfer Optimization Based on Historical Analysis and Real-Time Tuning

Engin Arslan, *Member, IEEE* and Tevfik Kosar, *Member, IEEE*

Abstract—Data-intensive scientific and commercial applications increasingly require frequent movement of large datasets from one site to the other(s). Despite growing network capacities, these data movements rarely achieve the promised data transfer rates of the underlying physical network due to poorly tuned data transfer protocols. Accurately and efficiently tuning the data transfer protocol parameters in a dynamically changing network environment is a major challenge and remains as an open research problem. In this paper, we present a novel dynamic parameter tuning algorithm based on historical data analysis and real-time background traffic probing, dubbed HARP. Most of the previous work in this area are solely based on real-time network probing or static parameter tuning, which either result in an excessive sampling overhead or fail to accurately predict the optimal transfer parameters. Combining historical data analysis with real-time sampling lets HARP tune the application-layer data transfer parameters accurately and efficiently to achieve close-to-optimal end-to-end data transfer throughput with very low overhead. Instead of one-time parameter estimation, HARP uses a feedback loop to adjust the parameter values to changing network conditions in real-time. Our experimental analyses over a variety of network settings show that HARP outperforms existing solutions by up to 50% in terms of the achieved data transfer throughput.

Index Terms—High performance networks, transfer tuning, application-layer optimization, network modeling, online tuning.

1 Introduction

As the trend towards increasingly data-intensive applications continues, developers and users need to invest significant effort into efficiently moving large datasets between distributed sites. Effective use of the available network bandwidth together with optimization of data transfer throughput have been critical for the end-to-end performance observed by most commercial and scientific applications. This is true despite multi-gigabit optical network offerings, since most users fail to obtain even a fraction of the theoretical speeds promised by existing networks due to issues such as sub-optimal end-system and network protocol tuning.

Most of the existing work on data transfer tuning and optimization is at the lower layers of the networking stack, including design of new transport protocols [1], [2], [3], [4], [5] as well as adapting and changing the existing transport protocols for better performance [6], [7]. At the application-layer, techniques have been proposed to keep the underlying transport protocol intact and tune it through parameters such as pipelining [8], [9], parallelism [6], [10], [11], [12], concurrency [13], [14], [15], and buffer size [16], [17], [18], [19] for improved performance. While significant performance gain can be achieved by tuning these application-layer transfer parameters [20], [21], [22], the optimal values of them vary depending on the characteristics of dataset (i.e., file size and the number of files), network (i.e., bandwidth, round-trip-time, and background traffic on network), and end-systems (i.e., file system and transfer protocol type). Thus, finding the best combination for these parameters is a challenging task.

- *E. Arslan is with University of Nevada, Reno. E-mail: earslan@unr.edu*
- *T. Kosar is with University at Buffalo, SUNY.*

Among these application-layer transfer parameters[1] pipelining helps in transferring multiple files back-to-back by eliminating the communication delay between client and servers, but the size of the transferred files must be small to benefit from it. Pipelining may even degrade the throughput when set to high values for large files. Instead, large files benefit from being divided into smaller blocks and transferred over multiple transfer channels. Similarly, both small and large files benefit from concurrency, meaning transferring multiple files simultaneously over multiple transfer channels. On the other hand, opening too many channels to transfer a single file (parallelism) or multiple files (concurrency) would degrade the throughput by causing network and storage congestion. Optimal values for these parameters depend on the many factors described above. In our previous work, we have developed heuristic-based dynamic optimization algorithms [20] to determine the best combination of these parameters.

In this paper, we present a novel transfer parameter tuning algorithm based on historical data analysis and real-time background traffic probing, called HARP. We use historical data to derive network specific models of transfer throughput based on the transfer parameters. Then, by running sample transfers, we capture the current load on the network which is used to prioritize the models based on their accuracy in estimating the current network traffic. Combining historical data analysis with real-time sampling enables our algorithms to tune the transfer parameters accurately and efficiently to achieve close-to-optimal end-to-end data transfer throughput with very low sampling overhead. Our experimental analyses over a variety of network settings show that HARP outperforms existing solutions by up

1. Application-layer transfer parameters will be referred as *transfer parameters* in the rest of the paper.

to 50% in terms of the achieved data transfer throughput. We also extend HARP with "online tuning" to monitor transfer throughput periodically and update values of the transfer parameters when the network conditions change, such as the background traffic increase and decrease. Online tuning is able to improve the data transfer throughput an additional 30-40% compared to HARP without online tuning.

The rest of this paper is organized as follows: Section II presents our system design and the proposed algorithm; Section III discusses the experimental evaluation of our model; Section IV describes the related work in this field; and Section V concludes the paper.

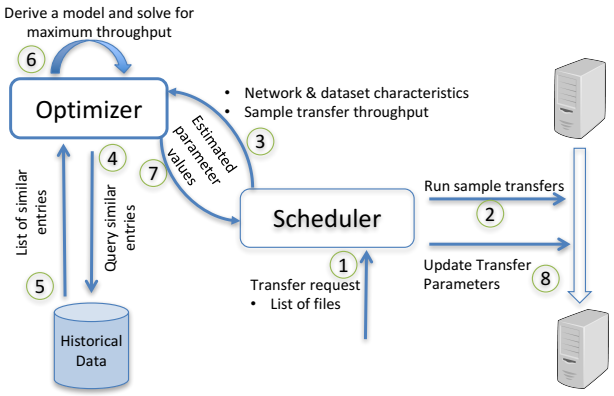


Fig. 1: Flow of operations in HARP.

2 Overview of HARP

HARP combines three transfer parameter tuning approaches: *i)* heuristics, *(ii)* real-time probing, and *(iii)* historical data analysis. Heuristic algorithms compute the values of the transfer parameters through calculations on the dataset and network metrics [20], [23]. For example, the value of pipelining is calculated by dividing bandwidth-delay-product (BDP) to average file size so that it will return large values for small files and small values for large files, which aligns with the purpose of pipelining [24]. However, heuristic approaches fail to capture end-system specific settings and dynamic components of networks, such as storage performances and background traffic. Real-time probing based approaches [22], [25] find the optimal values of the transfer parameters by running a sequence of sample transfers (probes) using different parameter values. They take a portion of the original dataset and transfer it with an initial parameter value (generally set to 1), followed by a series of sample transfers with increased values (2, 4, 8, etc.) until the observed throughput stops increasing. Although real-time probing has an advantage of capturing the instantaneous network load, too many sample transfers are needed to discover the best values of the transfer parameters. Finally, historical data based modeling solutions derive a model for transfer throughput based on dataset, network, and protocol metrics [26], [27]. In order to capture changes in the network load, they either rely on recent historical data [27] or run sample transfers [26].

HARP runs sample transfers similar to probing based methods, but the number of sample transfers is way less than it is in probing based algorithms. HARP benefits from heuristic solutions to determine the parameter values of sample transfers. Since poor choice of parameter values in the sample transfers may affect overall throughput, taking advantage of heuristic algorithms helps to alleviate the sampling overhead. Once the current network condition is observed, HARP models the transfer throughput similar to the solutions based on historical data analysis. While models driven by Kettimuthu et al. [27] and Nine et al. [26] require offline analysis and work well only for networks for which they are trained, HARP requires no prior data analysis and can be applied to different networks with the help of its extensible similarity detection algorithm. Furthermore, HARP can continuously monitor network conditions and adjust the parameter values if the background traffic changes, which is especially important for long running transfers.

HARP is composed of two major components, which are (1) Scheduler; and (2) Optimizer, as shown in Figure 1. When a data transfer request is submitted to Scheduler (step 1), it first categorizes files in the transfer request into clusters (a.k.a. file groups) based on the file size. Then, it runs one sample transfer for each file group to capture the load on the network (step 2) as well as the effect of the network load on the file groups. Once the sample transfer throughputs are obtained, Scheduler passes them to Optimizer along with the network settings (BW and RTT) and dataset characteristics (file metadata) (step 3) to determine similar entries in the historical data. Then, Optimizer identifies similar entries (step 4 and 5) and runs a regression analysis to derive a model that relates the transfer parameters to throughput. The derived model is then solved for the maximum transfer throughput and the corresponding parameter values are obtained (step 6). After the parameter values are found, parameter relaxation process is used to lower the values of parameters while keeping the estimated throughput in a reasonable range. Finally, Scheduler receives the estimated parameter values (step 7) and uses them to transfer the rest of the dataset (step 8).

The notations used in this paper are explained in Table 1.

Notation	Description
p	Parallelism; number of parallel connections for single file
cc	Concurrency; number of concurrent file transfers
pp	Pipelining; number of outstanding transfer commands
maxCC	Maximum allowed concurrency value for a transfer task
BW	Bandwidth of the network link
RTT	Round trip time
Buffer size	Maximum TCP buffer size defined at the end servers
ST	Sample transfer throughput
UT	Unit transfer throughput (when $cc = 1$)
SUT	Sum of unit transfer throughputs for all file clusters
T, Thr	Transfer throughput
	Relaxation threshold
	Residual error

TABLE 1: The notations used in this paper.

2.1 Transfer Scheduler

HARP's Scheduler is responsible for managing data transfer executions between source and destination end points. It first partitions files into clusters according to the file size (i.e., Tiny, Small, Medium, and Large) (line 3 of Algorithm 1). Then, it runs a sample transfer for each file cluster to

Algorithm 1 – HARP Scheduler

1: **function** TRANSFER(source,destination,BW,RTT)
2: $allFiles = getListofFilesFromSource()$
3: $clusters = partitionFiles(allFiles)$
4: **for each** cluster in clusters **do**
5: $(cc_0; p_0; pp_0) = heuristicParams(cluster; BW; RTT)$
6: $cluster:ST = runSampleTransfer(cluster; cc_0; p_0; pp_0)$
7: **end for**
8: $maxCC = 1$
9: $SUT = 0$
10: **for each** cluster in clusters **do**
11: $cluster:fcc_{est}; p_{est}; pp_{est}; UTg = runOptimizer(cluster:ST; BW; RTT; cluster:fileCount; cluster:avgFileSize)$
12: $SUT += cluster:UT$
13: $maxCC = max(maxCC; cluster:cc_{est})$
14: **end for**
15: **for each** cluster in clusters **do**
16: $cluster:weight = cluster:totalSize \quad \frac{SUT}{cluster:UT}$
17: $totalWeight += cluster:weight$
18: **end for**
19: **for each** cluster in clusters **do**
20: $cc^0 = min(cluster:cc_{est}; \lfloor maxCC \quad \frac{cluster:weight}{totalWeight} \rfloor)$
21: $transfer(cluster; cc^0; cluster:p_{est}; cluster:pp_{est})$. Async operation
22: **end for**
23: **end function**

determine the cluster's achievable throughput under current network load. Each sample transfer may move one or more files depending on file size. ST refers to sample transfer throughput for a file cluster. In order to minimize the overhead of the sample transfers (step 2 in Figure 1), Scheduler takes advantage of the heuristic algorithm [20] to determine the parameter values of the sample transfers. Although the heuristic algorithm may pick sub-optimal values, it generally perform better than using default or random values. We assess the cost of the sample transfers in detail in Section 2.3.

After real-time probing is completed, Scheduler passes throughput information to Optimizer along with the dataset characteristics and network settings (line 11). Optimizer calculates and returns the values of the protocol parameters ($cc_{est}; p_{est}; ppq_{est}$) along with unit throughput, UT (line 11). UT refers to throughput of a file cluster if it is run with concurrency value 1. It is used to determine the concurrency values of the clusters when they are transferred simultaneously. While Optimizer finds the optimal values assuming that each cluster will run separately, Scheduler prefers to run multiple clusters simultaneously to benefit from multi-cluster approach as presented in our earlier work [20]. As opposed to parallelism and pipelining, concurrency values estimated by Optimizer may need to be adapted to multi-cluster transfer scheme. Even though concurrency has significant impact on throughput (especially when parallel file systems are in use), it incurs the highest overhead at the end systems and network by creating many processes and data channels. Thus, it may not be desirable to open as many channels (concurrency) as each cluster asks. To overcome this inconsistency, Scheduler computes the maximum concurrency ($maxCC$) of all clusters (line 13) which is then distributed among the clusters based on their weights (line 20). Weight of a cluster is proportional to its size and inversely proportional to its unit throughput (line 16) such that clusters that are large in total size or inherently slow would receive more channels.

To give an example of channel distribution, assume that we have three file clusters in a given dataset and each cluster's size is 1 GB. Also assume that Optimizer returned (7,1,10,100), (4,3,1,200), and (3,5,0,400) for the three clusters as (cc,p,pp,UT) combination. Then, SUT will be 700 and $maxCC$ will be 7 (maximum of (7,4,3)). Based on weight calculation equation shown in line 16, 7x, 3.5x, and 1.75x weights will be assigned to clusters respectively since the cluster sizes are same in this case. Finally, when the maximum concurrency is distributed among clusters based on the respective weights, the first cluster will receive four concurrency, the second cluster will receive two concurrency and the last one will receive one concurrency. Since parallelism and pipelining values returned by Optimizer are used as is, the final transfer parameter values for the clusters will be (4,1,10), (2,3,1), and (1,5,0). Once the final parameter values are determined, Scheduler runs all three clusters concurrently.

2.1.1 Adaptive Sample Transfers

Two approaches have been proposed so far for running sample transfers. These are fixed data size [25] and fixed time duration [28] based sampling methods. In the first method, a fixed-size (e.g., 1 GB) portion of a original dataset is used to run sample transfers, however it requires an up-front work to identify the optimal data size [29] which may not be feasible for every transfer operation. Yet, intuitive choice of data size may lead to sub-optimal results as small values would cause low accuracy and large values would result in high delay. On the other hand, 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 run.

HARPuses an adaptive sample transfer method, in which Scheduler starts transferring an entire dataset and monitors instantaneous transfer throughput at certain intervals. If throughput of any two consecutive monitor intervals are closer than a threshold, then Scheduler exits sample transfer phase and takes the average of the two consecutive intervals' throughput as throughput of the sample transfer. Instead of using static values for threshold (which needs to be adapted to different network bandwidths), we define the threshold in terms of percentage of last monitor interval. That is, if the ratio of previous interval's throughput to current interval's throughput is less than a threshold value, then we assume that transfer throughput is converged.

We evaluated three values for the threshold in Figure 2(a) and Figure 2(b) for Small and Large file types under different monitor intervals (1, 3, 5 seconds) using Comet and Stampede end-systems on XSEDE. As the threshold increases, the sample transfer time shortens in exchange for higher error rates for both file types. 5% threshold with 3 seconds monitor interval is able to achieve around 10% error rate with less than 10 seconds delay. Compared to fixed-size sampling approach, it is 2-4X faster and more accurate. In Figure 2(c), we have tested 5% threshold under heavy background traffic. Although sample times and error rates increase a little bit, it is still able to achieve less than 15% error rate in 15 seconds which is almost twice better than fixed-size method. We validated that 5% threshold with 3 seconds monitor interval returns similar results in local area experiments but omitted due to space limitation.

(a) Small Files (b) Large Files (c) Heavy Background Traf c, 5% Threshold

Fig. 2: Adaptive sample transfers can achieve higher accuracy within shorter amount of time.

2.2 Optimizer

Optimizer aims to estimate the optimal values for the transfer parameters for an intended data transfer with the help of historical data and sample transfers. Thus, it heavily depends on the quality and quantity of the historical dataset to make the best decisions. Quality stands for how well the dataset captures variation in the network and dataset characteristics such as le size and background traf c. Quantity is important to detect outliers and derive accurate models.

2.2.1 Data Collection

We collected historical data on XSEDE [30], a production-level high-speed shared WAN, and DIDCLAB at UB, a dedicated LAN. The network and storage con gurations of the used systems are given in Table 2. Four different le sizes are used which are Tiny (varying from 1MB to 5MB with a total of 10GB), Small (15MB–30MB with a total of 20GB), Medium (50MB–200MB with a total of 40GB), and Large (1GB–5GB with a total of 98GB).

Specs	XSEDE Stampede-Gordon	DIDCLAB WS1-WS-2	EC2
Bandwidth (Gbps)	10	1	10
RTT (ms)	40	0.2	100
TCP Buffer Size (MB)	32	4	60
BDP (MB)	48	0.02	125
File System	Lustre	NFS	SAN
Max File System I/O Throughput (Mbps)	9600	720	2560

TABLE 2: Network speci cations of the test environments.

In order to observe the effects of the transfer parameters under different network loads, we tested the same parameter combinations under three network conditions; light, medium, and heavy background traf c. Although we can control background traf c in LAN experiments, we do not know the exact background traf c in a shared production network, XSEDE. However, we observed higher and more stable throughput values when we run transfers in the night hours. Thus, historical data is collected in the night hours of the day to minimize the effect of external load. Moreover, we repeated each entry at least ve times at different dates so that any outliers can be detected and neglected easily during the modeling phase. Different background traf cs are generated by running multiple memory-to-memory transfers in the background. Light background traf c means no synthetic traf c is generated by us. On the other hand, we created 8-stream and 32-stream memory-to-memory transfers for medium and heavy background traf c cases, respectively. Over a 12-week period, we collected statistics for 21K data transfers. Since we only kept metadata (i.e., number of les, average le size, date etc.) of these transfers, the amount of storage to store this data is around 4MB. Even though we did not experience a storage limitation in our experiments, one can de ne a time limit to remove old entries and keep the historical data size at a certain range in case of continuous data collection.

2.2.2 Data Filtering and Grouping

When Optimizer receives a request from Scheduler which consists of dataset characteristics, network settings, and sample transfer throughputs, it rst lters similar entries from the data store where the historical data is kept. Since it is possible that the data store may not have exactly matching entries for a given dataset/network setting, Optimizer uses a weighted cosine-similarity (shown in Equation 1) to measure similarities between the historical data entries and transfer tasks.

$$\cos(\ ) = \frac{\sum_{i=1}^{n} kA_i k kB_i k}{\sqrt{\sum_{i=1}^{n} kA_i^2 k}\ \sqrt{\sum_{i=1}^{n} kB_i^2 k}} \quad (1)$$

Cosine-similarity measures similarity between two non-zero vectors. In Equation 1, A and B are the two vectors and cosine-similarity calculates a similarity value based on closeness of vector elements in same positions. In the context of data transfers, the feature vector consists of dataset and network settings of the transfers e.g., bandwidth, round-trip-time, $\frac{BDP}{buffer\_size}$, le cluster type (Tiny, Small, etc.), le size, and le count. Since TCP buffer size must be equal to bandwidth delay product (BDP) to fully utilize network bandwidth, $\frac{BDP}{buffer\_size}$ is used to determine if the maximum allowed buffer size is suf cient. If not, parallelism can be used to overcome this limitation by opening multiple TCP connections, so aggregate buffer size can be equal to BDP. Even though some of the features are related to each other, such as the cluster type and the le size, we want to be as much speci c as possible in terms of similarity detection. For example, if BDP is 40 MB, then les with 1 MB and 1 KB sizes will be assigned to the Tiny cluster and les with 1 GB size will be assigned to the Large cluster according to our dataset partitioning method. On the other hand, if we just use le sizes to calculate similarities, les with 1 MB size will have same similarity value when compared to les

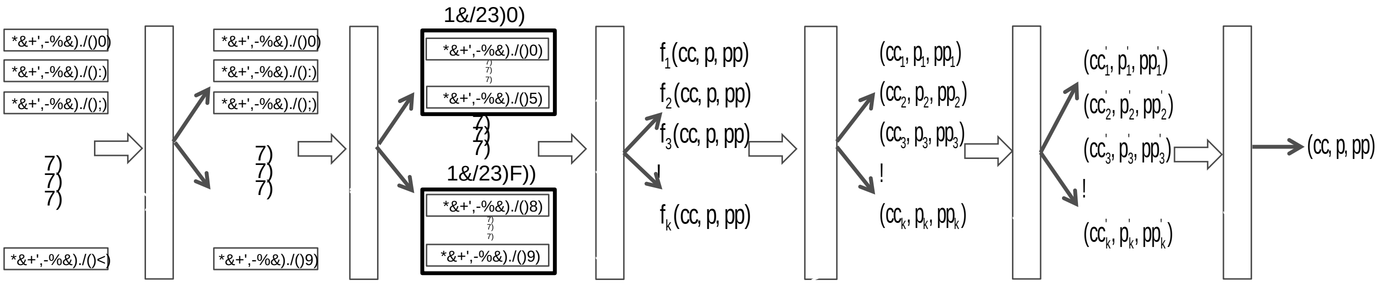


Fig. 3: Flow of operations in HARP's Optimizer.

with 1 KB and 1 GB sizes as both are in the same distance ($log(10^6) \quad log(10^3) = log(10^9) \quad log(10^6)$). However, we want files with 1 KB size to receive higher similarity values since they are assigned to the Tiny cluster as 1 MB files by dataset partitioning. To address such misclassification, we evaluate some features in multiple ways to achieve a more accurate similarity detection. Moreover, we normalize feature vectors of historical data entries to the keep values in close range, otherwise one feature may overweight the similarity value if its range is much larger than others. Finally, since each feature has a different impact on file transfer throughput, we assigned weights to them based on our initial effort to apply regression to the entire historical data. Although the accuracy of the regression was low, it gives a clue about the weight of each property on the achieved transfer throughput. Hence, we used (2,2,10,10,3,1) weight vector for the feature set (Bandwidth, RTT, $\frac{BDP}{buffersize}$, cluster type, file size and file count). Since there are other factors aside from dataset characteristics and network settings that affect the transfer throughput (e.g., background traffic and disk I/O performance), Optimizer runs an additional step of categorization using the results of the sample transfers, which is explained in Section 2.2.4.

Once Optimizer calculates the similarity value for each entry in the historical data by comparing them against the given transfer task, it picks the entries with similarity values larger than a threshold. We initialize the threshold value to 0.99 and decrease it until we have at least 6K entries. Since cosine-similarity does not consider background traffic, the selected entries may contain transfers with different background traffic. To identify entries with different background traffic, we grouped the set of entries that are not only same regarding to dataset and network characteristics but also collected at approximate times with the assumption that they are exposed to similar background traffic. Although it is possible that some of the entries collected at similar time periods are exposed to different background traffic due to transient traffic events, those will be identified as outliers and ignored in the modeling phase.

2.2.3 Regression Analysis and Nonlinear Equation Solver

After similar entries are grouped based on time information, entries within a group will have same network and dataset metrics but different transfer parameters. Hence, we can ignore dataset and network metrics and derive a model on this data that relates the transfer parameters to the transfer throughput as shown in Equation 2. Thr_i refers to the equation derived for i^{th} historical data group where $1 < i < k$ and k is equal to the total number

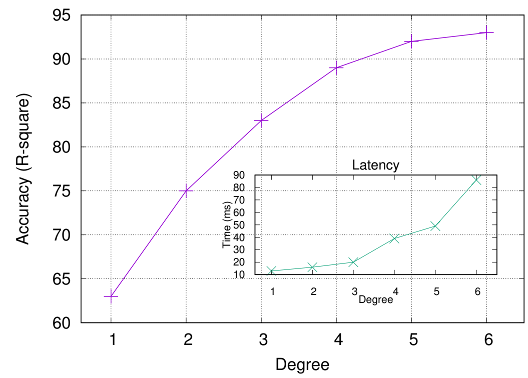


Fig. 4: Higher degree polynomial regressions return higher accuracy in return for longer computation time.

Model	R-square	Residual Variance
Linear Regression (LR)	0.503	740492.9
2^{nd} -degree Polynomial	0.632	547987.2
3^{rd} -degree Polynomial	0.672	488244.8
4^{th} -degree Polynomial	0.674	486482.1
Piecewise LR	0.618	356017.2
HARP	0.824	197223.3

TABLE 3: Goodness of fit for different models.

of historical data groups. By grouping entries based on the similarity in network/dataset metrics and background traffic, we can decrease the number of input parameters in the model which improves the regression accuracy. In order to validate the regression models, we divide each group into two subgroups as training (70%) and validation (30%) sets.

We compared several degrees of polynomial regression in terms of accuracy (on training data) and speed in Figure 4. According to the results, as we increase the degree of polynomial regression, the R^2 value increases as well as the time to compute the model. So, we start with degree two to compute regression for each group and increase the degree until R^2 for training and validation data goes above 0.7. If R^2 is still not larger than 0.7 for training and validation data of a group even for quartic (4^{th}-order) polynomial regression, then we classify the group as an outlier and ignore it. We observed that quadratic or cubic regressions satisfy the accuracy requirements for the most groups.

We compared the HARP's modeling approach against other regression techniques that try to fit a model on whole historical data in Table 3. While nonlinear polynomial regressions improve the accuracy of the fit over linear regression, they still fail to achieve R^2 higher than 0.7 and result in large residual variance due to not being able to incorporate background traffic into the model. While piecewise linear

regression decreases the residual variance to some extent compared to the nonlinear regressions, R^2 still stays less than 0.7. On the other hand, HARP is able to achieve much higher R^2 and signi cantly lower residual variance.

$$T_i = f_i(cc; p; pp) \quad (2)$$

After polynomial equations are derived for each group of historical data, $f f_1; f_2; :::; f_k g$, Optimizer evaluates them for the values used in the sample transfers to nd the estimated throughputs, $f T_1, T_2, ..., T_k g$ as shown in Equation 3. $_i$ refers to the difference between throughput estimated by f_i and obtained by the actual sample transfer, Thr_{act}. In order to prioritize historical data entries that are exposed to the background traf c similar to the instant one, Optimizer assigns weights to the equations based on their accuracy in estimating throughput of the sample transfers. One approach to determine the weights would be to use values that are reversely proportional to , however it would be vulnerable to small variations. For example, 1220 Mbps would have double weight over 1250 Mbps if sample transfer throughput, Thr_{act}, is 1230 Mbps while they both should have similar weight since the differences are very close. Thus, we classify the equations $f f_1; f_2; :::; f_k g$ into subgroups based on respective values using a density based clustering technique, DBScan. DBScan nds core points from the list $f_1; _1; ::: _k g$, and then adds the nearby points to the clusters of core points. Each equation in a subgroup is given a same weight and the weight of a subgroup is calculated from the list $f 2^0; 2^1; :::; 2^{t-1} g$ after subgroups are sorted in descending order based on values where t refers to the total subgroups extracted by DBScan. The subgroup weights play a signi cant role in (i) distinguishing historical data collected under different background traf cs and (ii) compensating similarity based ltering for possible misclassi cation due to unacknowledged factors of data transfers such as background traf c. We observed that DBScan generally returns 4-6 groups. While we only have three background traf c categories, additional groups are created since similarity analysis selects entries that are not only different in terms of background traf c but also in terms of network and dataset settings.

$$_i = T_{act} \quad f_i(cc_0; p_0; pp_0) \quad (3)$$

$$T_i = f_i(cc_i; p_i; pp_i) \quad (4)$$

After the weights of the subgroups are found, Optimizer nds the values of transfer parameters for each corresponding equation that returns the highest throughput using the non-linear optimization solver with Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [31]. BFGS seeks a stationary point $(cc_i; p_i; pp_i)$ using a hill scan optimization technique for the maximum throughput, $Tmax_i$, as shown in Equation 4.

2.2.4 Parameter Relaxation and Combiner

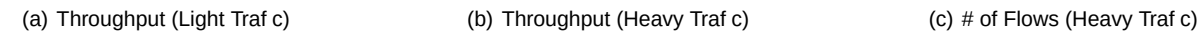
Once the corresponding parameter values for the maximum throughput are found for each equation f_i, $1 < i < k$, Optimizer runs the relaxation process during which the values of the parameters are lowered as long as the estimated throughput stays within a reasonable range as shown in Equation 5. For example, pipelining has either no or little contribution to the throughput of large les, but nonlinear equation solver may estimate large pipelining value for a marginal gain. However, small improvement in the transfer throughput may cause a signi cant increase in power consumption at the end servers and network equipment [32], [33]. In our earlier work [28], we have shown that one can achieve a similar transfer throughput while consuming up to 30% less energy by tuning the transfer parameters. Moreover, using very large concurrency and parallelism values would lead to a signi cant resource allocation for a single transfer task (by creating many threads and processes) which is undesirable in shared networks. Hence, the relaxation process tries to identify the sweet spot for the parameter values in which a reasonable throughput can be achieved with a minimal system overhead.

$$\begin{aligned} pp_i^0 \quad pp_i \; j \; f_i(cc_i; p_i; pp_i^0) = T max_i^0 \quad _{pp} \quad T max_i \\ p_i^0 \quad p_i \; j \; f_i(cc_i; p_i^0; pp_i^0) = T max_i^{00} \quad _p \quad T max_i^0 \\ cc_i^0 \quad cc_i \; j \; f_i(cc_i^0; p_i^0; pp_i^0) = T max_i^{000} \quad _{cc} \quad T max_i^{00} \end{aligned} \quad (5)$$

$$cc_{avg} = \sum_{i=1}^{k} \frac{cc_i^0 \; w_i}{w_{total}} \qquad p_{avg} = \sum_{i=1}^{k} \frac{p_i^0 \; w_i}{w_{total}}$$
$$pp_{avg} = \sum_{i=1}^{k} \frac{pp_i^0 \; w_i}{w_{total}} \quad (6)$$

In the relaxation phase, Optimizer evaluates smaller values for each parameter until the estimated throughput is larger than certain percentage of the original throughput. Let's assume values $(32; 20; 24)$ are calculated as optimal values for concurrency, parallelism and pipelining for an equation f such that $f(32; 20; 24) = Tmax$. The relaxation process evaluates smaller values for pipelining starting from 1, while keeping parallelism and pipelining the same, until the projected throughput, $Tmax^0$, becomes equal or slightly larger than the certain percentage of the initially estimated throughput ($Tmax^0$ $Tmax$). For example, if $f(32; 20; 24) = 5600$ and $_{pp} = 0:9$, then the relaxation process will search for the minimum pipelining value pp^0 that satis es $f(32; 20; pp^0) > 5600 \quad 0:9$. We observed that $= 0:99$ works well enough for pipelining to detect high pipelining value estimations by non-linear programming solver for a small throughput gain.

We evaluated different values for concurrency and parallelism while keeping the threshold for pipelining at 0.99 as shown in Figures 5. In the Figures 5(a) and 5(b), y-axis represents transfer throughput which is proportional to maximum throughput achieved among all relaxation values. X-axis shows the values for concurrency and parallelism. Although the effect of using small threshold values for parallelism does not seem to be obvious in the Light Traf c case, it is easily noticeable in the Heavy Traf c case. Moreover, while parallelism has little impact on throughput of Small le types under light background traf c, its impact becomes signi cant as background traf c increases as shown in Figure 5(b). Figure 5(c) shows the number of network ows opened for different relaxation thresholds. Although disabling relaxation process or using large threshold values would yield high transfer throughput, it would create too many network connections and processes which would increase the load on network and



(a) Throughput (Light Traf c) (b) Throughput (Heavy Traf c) (c) # of Flows (Heavy Traf c)

Fig. 5: Effect of different relaxation ratios for concurrency-parallelism pair for different le sizes in WAN (Stampede-Comet).

T_H Speed-up (%)	T_S Slowdown (%)	Min. Cluster Size (D) (Thr_0)
10	50	90
10	30	60
10	10	30
30	50	40
30	30	30
30	10	20
50	50	30
50	30	24
50	10	18

TABLE 4: Minimum cluster size for HARP to pay off.

end system devices. Thus, we aimed to nd a relaxation threshold that would achieve a reasonable throughput without causing too much overhead on network devices and servers. Hence, we picked 0.7-0.7 threshold combination for concurrency and parallelism which achieved 60% or more of maximum throughput while keeping the number of ows and processes at a reasonable scale. We also con rmed that 0.7-0.7 threshold combination works well in local area experiments as well.

2.3 Cost Analysis of HARP

HARP runs sample transfers and applies data modeling in the real-time so it comes with an overhead. To minimize the overhead, we overlap the transfer sampling process with the optimization process at the best effort. Instead of waiting for each le cluster's sample transfer to be completed, we run Optimizer for a cluster as soon as its sample transfer is completed so that Scheduler and Optimizer can operate simultaneously. For example, once Scheduler nishes the sample transfer for the Small cluster, it starts running the sample transfer for the Medium cluster. While the sample transfer for the Medium cluster is running, it passes the sample transfer throughput of the Small cluster to Optimizer so that it can compute and return the transfer parameter values in the meantime. Since Optimizer can nish the calculations in around 2-3 seconds for each cluster, the bottleneck in the pipelined process becomes the sample transfers. Then, the overall cost boils down to the cost of the sample transfers plus running Optimizer for the last cluster.

$$t_0 = \frac{D}{Thr_0} \tag{7}$$

$$t_H = \frac{D \quad D_S}{Thr_H} + \frac{D_S}{Thr_S} + c \tag{8}$$

$$t_H = \frac{D \quad (15 \quad Thr_S)}{Thr_H} + 15 + c \tag{9}$$

Equation 7 shows the duration of a data transfer when HARP is not used. D refers to the total data size and Thr_0 refers to the achieved throughput. When HARP is used, the duration of the the data transfer is determined by Equation 8 in which D_S refers to the data size of a sample transfer and Thr_H and Thr_S refers to the throughput values obtained in the sample transfer and actual transfer, respectively. c refers to the cost of Optimizer for running the optimization process for the last le cluster. As explained in Section 2.1.1, when the adaptive sampling approach is used with 5% threshold and 3-second monitoring interval, the sampling process nishes in 15 seconds in the worst case scenario. Thus, $\frac{D_S}{Thr_S}$ becomes 15 and sample transfer data size, D_S, becomes $15 \quad Thr_S$. Hence, t_H reduces to Equation 9.

In Table 4, we calculated the minimum data size D values needed for HARP to amortize the cost it induces (aka $t_0 = t_H$) under the different Thr_H speed-ups and Thr_S slowdowns. T_H Speed-up column represents $\frac{Thr_H \quad Thr_0}{Thr_0}$ which stands for the throughput gain when HARP is used. Although the gain will be much higher when HARP is compared with Globus Online [23] and PCP [25], we compared HARP against the heuristic algorithm (ProMC) we proposed in the previous work [20] which outperforms Globus Online and PCP by a signi cant margin. T_S Slowdown column represents the ratio of transfer throughput decrease during the sample transfers. Since we transfer each le cluster separately in the sampling phase, the throughput is generally smaller than ProMC which transfers multiple clusters simultaneously. For example 30% slowdown means $Thr_S = (1 \quad 0{:}3) \quad Thr_0$. Min Cluster Sizecolumn is represented in Thr_0 since Thr_H and Thr_S values also measured in Thr_0.

In the worst case scenario, the gain of HARP is 10% and the sample transfers are 50% slower than Thr_0, the minimum le cluster size that HARP pays off the cost is $90 \quad Thr_0$. It will become 67 GB when Thr_0 is 6 Gbps. The minimum cluster size to bene t from HARP reduces as the throughput gain increases or the sample transfer slowdown reduces. Our observations on the tests we ran in XSEDE, AWS and DIDCLAB networks reveals that slowdown ratios mostly stay less than 50% and the gain ratio ranges from 10% to 80%. Hence, the results we present in Section 3 show that HARP mostly outperforms the heuristic

Fig. 6: Single type le transfers between Stampede (TACC) and Gordon (SDSC) on XSEDE.

algorithms under the light traf c and improves the overall throughput signi cantly under the medium and heavy background traf c cases. Furthermore, we explore an online tuning approach in Section 3.2 which eliminates the delay for running optimization process for the last cluster (c) as well so that the cost of HARP reduces even further.

3 Experimental Analysis

We compared HARP against Globus Online [23], Single Cluster [20], ProActive Multi-Cluster [20], and PCP [25]. Globus Online (GO) separates transfer datasets into clusters based on le size and uses prede ned values for the transfer parameter values for each cluster. Single Cluster (SC) algorithm, similarly, separates the dataset based on le size and transfers them one by one using the parameter values found by heuristics. ProActive Multi-Cluster (ProMC) creates le clusters and determines the values of the transfer parameters similar to SC, but instead of transferring the clusters one by one, it transfers all of them at the same time in order to minimize the effect of small clusters on overall transfer throughput. PCP employs a divide-and-transfer approach similar to SC and GO. It determines the transfer parameter values by running several sample transfers. Similar to SC and Globus Online, it transfers clusters one by one. One chunk at-a-time policy coupled with too many sample transfers degrades PCP's overall performance signi cantly.

We tested HARP in the networks we had collected historical data (e.g., Stampede-Gordon and WS1-WS2) as well in the networks we had not (e.g., Gordon-Stampede, Bluewaters-Stampede, SuperMic-Bridges, and Cloud). Datasets used in the experiments are different from the ones used in the historical data collection process. While datasets in data collection process are homogeneous (e.g., 10000 x 1 MB les are used to represent the Tiny cluster), the experiment datasets are heterogeneous (i.e., contain both small and large les). Additionally, le sizes in a le type are determined randomly such that we do not assume any le size distribution within a group as well. All experiments are repeated at least ve times.

We rst transferred datasets that are composed of only one type of le size; either all large or all small les. Figure 6 shows the comparison of GO, SC, ProMC, PCP, and HARP. The size of the datasets are 45 GB and 92 GB for small and large les, respectively. While SC and ProMC achieve a similar throughput for small le transfers, they differ in large le transfers because of the way SC calculates the concurrency level of a cluster. While ProMC uses the maximum allowed concurrency value (in this case it is 10), SC may prefer using less since it determines the value of concurrency by taking the minimum of what it calculates and what is given by the user as an upper bound. It decides to use the concurrency value 2 for large les, thus yields lower throughput than ProMC. PCP performs worse than SC for small les which is due to the overwhelming effect of the sample transfers. While, HARP outperforms ProMC by around 25% for small les, it obtains 5% less throughput than ProMC for medium and large les. Digging into details, we found out that Optimizer estimates the concurrency value in 10-13 range for large les which is close to what ProMC is set to run in this experiment. Although HARP yields a bit higher throughput after Optimizer estimates concurrency, due to sampling and optimization overhead, it falls slightly behind ProMC. It is worth to note that, while ProMC performs close to HARP in this experiment, one has to know what concurrency value to pass to ProMC which requires a domain expertise.

Figures 7(b) and 7(c) are the results obtained in the networks where the historical dataset contains exact matching entries. We have tested the algorithms under three different network loads (light, medium, and high). In both networks, the throughput of the most algorithms decreased by 50-300% as the network load increases.

Algorithms that transfer one cluster at a time (GO, PCP, and SC) exhibit poor performance on XSEDE because their overall performance is pulled down by the throughput of small le transfers. GO, PCP, and SC achieve less than 3 Gbps under light background traf c. On the other hand, multi-cluster algorithms (ProMC, and HARP) are able to deliver around 7 Gbps. As discussed in Section 2.3, HARP requires data size to be greater than certain amount to outperform the ProMC. The size of dataset used in Figure 7(b) was 130 GB which is only enough to cover the overhead imposed by HARP. When we increase the dataset size to 260 GB, throughput of HARP reached to 8 Gbps as its overhead is alleviated by the increase in transfer duration. We omitted this comparison due to space limitation. Unlike light background traf c case, HARP gains 36% and 48% more throughput than ProMC even for smaller dataset size (130 GB) under the medium and heavy background traf c conditions by taking the advantage of the historical transfer information. The reason why ProMC performs worse as the network load varies is that its parameter estimation approach is oblivious to network traf c which fails as the network condition differs from its expectation. One may claim that ProMC can perceive the network load by running sample transfers, however, without historical data, it cannot interpret the probing results. A simple way to interpret probing is done by PCP algorithm which runs several probings and increments the parameter values until the probing throughput decreases. However, results show that it fails to achieve high transfer throughput since too many probings are required to nd the optimal values, which negatively affect its overall performance. The throughput of SC is dropped drastically from 3.2 Gbps to 750 Mbps as the network load increases. Similarly, throughput of GO suffers

(a) Stampede - Gordon (XSEDE) (b) Gordon-Stampede (XSEDE) (c) WS1-WS2 (DIDCLAB)

Fig. 7: HARP adapts the transfer parameters to varying background traf c and yields higher transfer throughput.

Fig. 8: HARP outperforms ProMC in all traf c types in Cloud experiments. The difference goes up to 30% under heavy background traf c.

Fig. 9: HARP achieves 23% to 30% speed-up over ProMC in Dark Energy Survey data transfer under medium and heavy background traf c.

signi cantly. PCP is able to adapt the transfer parameters values and outperform SC and GO under the high network loads. However, the overhead of sampling and “one cluster at a time” policy limits its overall performance to less than 1 Gbps in congested network conditions.

We picked two DIDCLAB servers (WS-1 and WS-2) to test HARP in a LAN with a single-disk storage subsystem. Figure 7(c) presents the performance comparison of different algorithms. When the background traf c is light, SC performs better than ProMC since opening too many processes (aka concurrency) to execute disk I/O operations deteriorates the I/O throughput while not improving network throughput. However, as background traf c increases, opening many connections helps to obtain higher network throughput. HARP yields 650 Mbps throughput under the light background traf c which is limited by the storage performance. On the contrary, network throughput becomes the bottleneck when the network is highly congested. Hence, HARP achieves a higher end to end transfer throughput by adapting the transfer parameter values to varying network conditions.

Since heuristic algorithms do not take storage performance metrics into account when calculating protocol parameters, they fail to perform well when disk I/O throughput decreases as the number of concurrent le transfers increases. Hence, HARP can outperform heuristic algorithms with the help of historical data even when there is a relatively small dataset. When there is no background traf c, HARP achieves 13% higher throughput than ProMC. As the network load increases, HARP obtains 47% and 37% more than ProMC for the medium and high background traf c cases.

In order to test the effectiveness of HARP for networks that have no matching entries in the historical data, we run experiments on XSEDE and Amazon EC-2. Although the same pair of servers are used in XSEDE experiments, we have transferred dataset in a reverse direction of historical data entries. Namely, historical data has the logs for the transfers that are sourced from Stampede and destined to Gordon. In this experiment, Gordon is used as a source and Stampede became the destination. Although it may seem identical to the Stampede-Gordon transfers, the results show that the maximum achievable throughput is different than the Stampede-Gordon transfers. This is because (i) these sites exhibit different performance results for disk read and write operations and (ii) available network bandwidth is not the same in both directions. Hence, the Gordon-Stampede transfers can be used to show how HARP performs on the networks that the historical data have similar but not exactly the same entries.

While the Stampede-Gordon transfers experienced 50-300% as network load increases, the impact stayed around 10-100% for the Gordon-Stampede transfers as shown in Figure 7(a). HARP and PCP were affected the least by the increased network traf c compared to the heuristics since they probe network status at the beginning of transfer and picks the parameter values accordingly. For example, HARP obtained 13% more throughput than ProMC under the light background traf c. The difference increased to 24% as the throughput of ProMC dropped by 24% un-

der the heavy background traf c while the throughput of HARP only dropped by 11%.

Finally, we tested HARP in Amazon EC-2 where we used two c3.8xlarge instances and Provisioned IOPS EBS storage volumes with disk read/write speed of 320 MB/s as shown in Table 2. However, we observed 265 MB/s maximum disk throughput. Similar to XSEDE experiments, the algorithms that transfer multiple clusters simultaneously (ProMC and HARP) performed better than the single cluster algorithms at all background traf c loads as shown in Figure 8. Similar to the Gordon-Stampede experiments, the results are not affected as much as Stampede-Gordon experiments by the increased network loads since the transfer performances are again limited by disk I/O throughput.

Due to the similarities in network settings, cosine-similarity favors XSEDE transfers over LAN transfers when ltering similar entries in historical data. Since the EC-2 experiments resemble to XSEDE network in the context of yielding higher disk I/O throughput as the number of concurrent transfers increases and having a smaller buffer size than BDP, the models derived using XSEDE entries work well in the EC-2 experiments. Hence, HARP outperforms ProMC by 5% in the light background traf c case. The difference reaches to 34% and 32% under the medium and high network loads as ProMC fails to adapt the transfer parameter values to varying system load.

Optimizing Dark Energy Survey Data Transfer

Dark Energy Survey [34] captures pictures of space to probe the dynamics of the expansion of the Universe and the growth of large-scale structure. It collects 200-300 GB of data each day which is transferred from observatory in Chile to collaborating research institutions in America and Europe. We took one day worth of data which consists of 428 les, sizes range from 270 MB to 730 MB, and transferred from Stampede (TACC) and Gordon (SDSC) under different background traf c conditions.

Figure 9 shows the performance results for Globus Online, heuristics (SC and ProMC), and HARP. SC and Globus Online yileds 2-3x less throughput than ProMC and HARP due to the underestimation of transfer parameters. ProMC and HARP perform similar under the light background traf c as they both estimate using similar parameter values and obtain close-to-maximum transfer throughput. However, as the background traf c increases ProMC is again affected signi cantly and obtains 23% and 30% less throughput than HARP.

3.1 The accuracy of the Model

Table 5 shows average values of the parameters for transfers in Wide Area and Local Area networks under Light (L) and Medium (M) background traf c conditions. Since the transfer parameters have different impact on different le sizes, we have gathered the transfer parameters for each le type separately.

HARP is able to differentiate WAN and LAN transfers by estimating high concurrency values for the WAN transfers and low concurrency values for the LAN transfers. It also calculates high concurrency values for the small le types and high parallelism values for the large le types. Although Nonlinear Equation Solver, in general, tends to estimate high concurrency and parallelism values, the relaxation process decreases them a bit to avoid overloading network and end systems. In addition, it picks higher parallelism values when networks are congested in order to increase its bandwidth share by creating more connections.

After the ltering process of Optimizer selects similar entries from historical data and the grouping operation categorizes them, we allot 30% of each group as validation data and use the rest to train the model. To measure the correctness of the derived model, we rst calculate the optimal parameter values using training data. Assume that it returns $(cc_{training}; p_{training}; pp_{training})$ for corresponding throughput $Thr_{training}$. Then, we use validation data and apply polynomial regression to derive model, f_{test}, and nd its optimal parameter values and estimated throughput, Thr_{test}. Then, instead of directly comparing throughputs $Thr_{training}$ and Thr_{test}, we calculate throughput $Thr_{projected} = f_{test}(cc_{training}; p_{training}; pp_{training})$ in order to compute how well the training data parameters do in validation data model. Direct throughput comparison may not be accurate because test data and training data might have been exposed to different background traf c, thus maximum throughput of the two model could be different even if the optimal parameters are the same. So, we calculate the validation accuracy as $\frac{|Thr_{test} - Thr_{projected}|}{Thr_{projected}}$. We also listed estimation accuracy which measures closeness of estimated throughput to the actual transfer throughput. Estimation accuracy might not be a good metric to judge the model since actual throughput of a network may change over time even though the optimal parameters stays the same.

Validation accuracy of HARP is always above 85% which indicates success of regression analysis in modeling transfer throughput. While the estimation accuracy is more stable and comparatively high in LAN, it is worse in the WAN experiments since it is an uncontrolled environment so resource capacities might have changed between data collection and experimenting periods. Looking into deeper why estimation accuracy is 41% for Tiny le type in WAN experiment, we have discovered that, while the maximum throughput of the Tiny cluster in historical data never reaches beyond 4 Gbps, we observed 5.5 Gbps in test experiments. Thus, the estimation accuracy highly depends on consistency of historical data with current network status. This can easily be handled by logging every real time transfer so that the historical data always keep up-to-date information.

3.2 Online Tuning

Since some transfers can take tens of minutes, hours or even days, it is inevitable that background traf c changes while these transfer are running. Figure 10 shows achievable memory-to-memory transfer throughput between two XSEDE site pairs for a week period. Memory-to memory transfers help to eliminate the effect of le system related congestion on transfer throughput. So, the only factor that may cause throughput variation is background traf c. We used two parallel streams for BlueWaters-Comet transfer and four parallel streams in SuperMIC-Bridges in order to

	Stampede-Gordon (WAN)								WS1-WS2 (LAN)							
File Type	Tiny		Small		Medium		Large		Tiny		Small		Medium		Large	
Traf c	L	M	L	M	L	M	L	M	L	M	L	M	L	M	L	M
Concurrency	24	25	22	22	10	10	12	10	6	4	2	1	1	1	2	1
Parallelism	0	0	11	10	18	19	15	19	3	6	7	11	9	13	10	9
Pipelining	5	4	0	0	0	0	0	1	2	1	1	2	1	2	2	1
Validation Accuracy (%)	95	96	94	93	90	85	93	91	90	86	93	91	90	88	88	88
Estimation Accuracy (%)	41	62	78	77	81	73	85	86	84	79	91	76	90	91	87	86

TABLE 5: Sample transfer metrics and accuracy values of HARP under Light (L) and Medium (M) background traf cs.

(a) BlueWaters-Comet (b) SuperMIC-Bridges

Fig. 10: Online Tuning detects increasing background traf c and increases the number of ows for higher throughput

overcome buffer size limitations. Although both transfers are run simultaneously, since they do not share a common link, they do not compete with each other. In Figure 10(a), it can be seen that, while throughput is more or less stable in short time periods (within hours), it varies signi cantly over time. Similar behavior can be observed in Figure 10(b) as throughput uctuates between 120 Mbps and 410 Mbps. While some days' traf c pattern is similar, there is no de nitive pattern to predict the future traf c. Hence, one-time sampling at the beginning of the transfer will not work well for long running transfers. Therefore, we extended HARP with online tuning which periodically monitors the transfer throughput and calculates new parameter values based on the observed background traf c.

Online Tuning also helps to eliminate the need for a special sampling phase since we can start with some initial values until new parameter values are calculated by Optimizer. Eliminating the sampling phase mitigates the connection setup/teardown cost that sample transfers induce. Also, Scheduler does not have to wait for Optimizer to nish its operation as they can work simultaneously. Online tuning lets Scheduler measure the transfer throughput in certain monitor intervals and conveys it to Optimizer. Optimizer then can run the modeling and parameter estimation operations to estimate new values for the parameters. Meanwhile, Scheduler does not have to wait for Optimizer, since the results of Optimizer can be applied in the next interval. That is, while Scheduler is in interval MI_{i+1}, Optimizer can calculate the parameter values based on previous interval MI_1 and newly proposed values can be applied at the end of the interval MI_{i+1}.

Since it is possible that some throughput variations may happen even when background traf c is the same, we consider last k monitor intervals when making a decision on whether or not to update the parameter values to avoid transient uctuations. After at least k periods have passed and Optimizer consistently suggests using different parameter values, then Scheduler updates the parameter values. Among the three transfer parameters, pipelining is the easiest parameter to update the value as it does not require a connection setup/teardown. On the other hand, concurrency means establishing a new connection between source and destination servers which may take a couple of seconds due to slow authentication process. While parallelism should not require a new connection, the current implementation lets its value to be set only when connection is rst established. Hence, we have to close an existing connection and establish a new one with an updated parallelism value. Since connection setup/teardown is a costly operation, we update the concurrency and parallelism values only if there is at least two differences between old and new values such that an expected gain pays off the induced cost. For example, if we are currently running a transfer with a concurrency level 4 and Optimizer returns 5 in the last k periods, then Scheduler will not apply it. While the right value for k may affect the accuracy and stability of online tuning, we observed that $k = 4$ works well in the experiments as it is large enough to avoid instant

(a) Small Files (b) Large Files

Fig. 11: Online Tuning detects increasing background traf c and increases the number of ows for higher throughput

(a) Small Files (b) Large Files

Fig. 12: Online Tuning detects decreasing background traf c and decreases the number of ows to minimize network overhead

throughput variations and small enough to catch prolonged background traf c changes.

Figure 11 and 12 show the comparison of instantaneous throughput of HARP [35] and HARP with online tuning (HARP w/ OT) for transfers in the XSEDE network. In order to emulate dynamic network conditions, we transitioned background traf c from: light to heavy (Figure 11) and heavy to light (Figure 12) some time after transfers started. Transitions are marked with a solid vertical line. “Heavy Background Traf c Starts” and “Heavy Background Traf c Ends” texts mark the start and the end of heavy background traf c which is controlled manually. We also tracked the number of active TCP ows used in transfers to have an idea about how much overhead is imposed to network in different scenarios. Flow numbers are calculated by multiplying concurrency and parallelism values.

Although we smoothed the instant throughput, it can be seen in the Figure 11(a) that HARP's throughput decreases a bit around the 30^{th} second. This is because of the the way HARP is designed to operate. It rst runs the sample transfers and conveys the results to Optimizer Scheduler sits idle while Optimizer is working on estimating the optimal values. This idle time can be eliminated with the help of online tuning since we do not need a separate sampling phase anymore. Rather, Optimizer and Scheduler can work simultaneously and Scheduler can apply any proposed changes in the next interval.

Since online tuning requires at least four consecutive intervals of consistent input from Optimizer to apply proposed changes, there is a delay between the start of heavy background traf c and OT's reaction as in Figure 11(a) and 11(b). While Scheduler does not require the same exact values to be returned by Optimizer in the four consecutive periods, it expects to receive consistently larger or smaller values (compared to currently used ones) for a parameter in order to make sure that higher or lower values are not caused by a transient traf c.

HARP without OT estimates parameter values only once right after the sample transfer and keeps the same values throughout the transfer. So, the number of network ows changes once and only after sample transfers. On the other hand, when OT is enabled, HARP can react to varying background traf c as many times as possible so the number of ows may change multiple times. By adapting parameter values to varying background traf c conditions HARP with OT achieves 30-40% higher overall throughput and completes the transfers 160-185 seconds faster.

If HARP starts to run when background traf c is heavy, it estimates higher parameter values to obtain high transfer throughput as shown in Figure 12. However, as background traf c decreases, the bene t of using high parameter values starts disappearing. For example, running the transfer with large parameter values under low background traf c achieves around 2-5% more throughput for large les (Figure 12(b)) while it causes 15% decrease for small les (Figure 12(a)). Decreasing the number of ows as background traf c transitions from heavy to light leads to a higher instantaneous transfer throughput for small les due to parallelism value. When background traf c is heavy, large parallelism values help to achieve higher transfer throughput by increasing its share in the network bandwidth. On the contrary, it causes a decrease in throughput when background traf c is low since the overhead of using parallelism is not accommodated by a throughput increase when the current throughput is already high. Thus, HARP with OT outperforms HARP despite using smaller number of ows.

As a result, online tuning does not only help to keep the number of flows minimal in return for a small performance sacrifice for large files, but also obtains higher throughput by using a smaller parallelism level for small files.

4 Related Work

Liu et al. [15] developed a tool which optimizes multiple transfers by opening multiple GridFTP threads. The tool increases the number of concurrent flows up to the point where the transfer performance degrades. Their work only focuses on concurrent file transfers, and other transfer parameters are not considered. Globus Online [23] offers fire-and-forget file transfers through thin clients over the Internet. The developers mention that they set the pipelining, parallelism, and concurrency parameters to specific 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 consider real-time background traffic conditions.

Other approaches aim to improve the transfer throughput by opening flows over multiple paths between end-systems [36], however there are cases where individual data flows fail to achieve optimal throughput because of the end-system bottlenecks. Several others propose solutions that improve utilization of a single path by means of parallel streams [22], [37], [38], pipelining [24], and concurrent transfers [13], [14]. Although using parallelism, pipelining, and concurrency may improve throughput in certain cases, an optimization algorithm should also consider system configuration, since the end-systems may present factors (e.g., low disk I/O speeds or over-tasked CPUs) which can introduce bottlenecks.

Yildirim et al. [39], Yin et al. [40], and Kim et al. [41] proposed highly-accurate predictive models solely based on real-time probing which would require as few as three sampling points to provide very accurate predictions for the parallel stream number giving the highest transfer throughput. These models have proved to provide higher accuracy compared to existing similar models in the literature [38], [42]. Later, Yildirim et al. presented the PCP algorithm to dynamically tune parameter values of data transfer [25]. PCP categorizes files into three groups based on file size (small, medium, and large) and then run sample transfer for each file group to determine parameter values that would return higher transfer throughput. Although PCP does not require historical data to operate and it can adapt itself to varying network conditions, it requires too many sample transfers to adapt to the dynamically changing network environment.

In our earlier work, we proposed heuristic algorithms [20] to determine the best parameter combination by using network and dataset characteristics. Nine et al. developed ANN+OT [26] which uses historical data to derive model that relates transfer metrics to transfer throughput. It then runs real-time probing in order to capture the current network status. None of the existing approaches can perform accurate optimization for mixed datasets, which is one of the primary contributions of HARP. HARP also performs combined offline and online optimization, utilizing both historical data analysis and real-time probing, taking the changes in the background traffic into consideration.

5 Conclusions

In this paper, we presented a dynamic end-to-end data transfer optimization algorithm based on historical data analysis and real-time background traffic probing, called HARP. Most of the existing work in this area is solely based on real-time network probing or static parameter tuning, which either cause too much sampling overhead or fail to accurately predict the optimal transfer parameters. Combining historical data analysis with real-time sampling enables HARP to tune the application-layer data transfer parameters accurately and efficiently to achieve close-to-optimal end-to-end data transfer throughput with very low overhead. HARP uses historical data to derive network specific models of transfer throughput based on protocol parameters. Then by running sample transfers, we capture current load on the network which is fed into these models to increase the accuracy of our predictive modeling. Our experimental analyses over a variety of network settings show that HARP outperforms existing solutions by up to 50% in terms of the achieved throughput. We also extended HARP with online tuning to be able to react to the varying network conditions by adapting the transfer parameter values. We observed that online tuning improves HARP's performance by up to 40% under unpredictable and varying background traffic conditions.

Acknowledgments

This project is in part sponsored by the National Science Foundation (NSF) under award number OAC-1724898. It uses XSEDE resources for some of the experiments, which is supported by NSF under award number ACI-1548562.

References

[1] 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.

[2] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "Pcc: Re-architecting congestion control for consistent high performance." in NSDI, 2015, pp. 395–408.

[3] 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.

[4] J. Crowcroft and P. Oechslin, "Differentiated end-to-end internet services using a weighted proportional fair sharing tcp," SIGCOMM Comput. Commun. Rev. vol. 28, no. 3, Jul. 1998.

[5] G. Kola and M. K. Vernon, "Target bandwidth sharing using endhost measures," Perform. Eval. vol. 64, no. 9-12, Oct. 2007.

[6] J. Lee, D. Gunter, B. Tierney, B. Allcock, J. Bester, J. Bresnahan, and S. Tuecke, "Applied techniques for high bandwidth data transfers across wide area networks," in International Conference on Computing in High Energy and Nuclear Physics, April 2001.

[7] G. Kola, T. Kosar, and M. Livny, "Run-time adaptation of grid data-placement jobs," Scalable Computing: Practice and Experience, vol. 6, no. 3, pp. 33–43, September 2005.

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

[9] 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, 2002.

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