HARP: Predictive Transfer Optimization Based on Historical Analysis and Real-time Probing

Engin Arslan, Kemal Guner, Tevfik Kosar
Department of Computer Science & Engineering
University at Buffalo (SUNY), Buffalo, New York 14260
{enginars, kemalgne, tkosar}@buffalo.edu

***Abstract*—Increasingly data-intensive scientific and commercial applications require frequent movement of large datasets from one site to the other. Despite the growing capacity of the networking capacity, these data movements rarely achieve the promised data transfer rates of the underlying physical network due to the poorly tuned data transfer protocols. Accurately and efficiently tuning the data transfer protocol parameters in a dynamically changing network environment is a big challenge and still an open research problem. In this paper, we present predictive end-to-end data transfer optimization algorithms based on historical data analysis and real-time background traffic probing, dubbed HARP. Most of the existing work in this area is solely based on real time network probing, which either cause too much sampling overhead or fail to accurately predict the correct transfer parameters. Combining historical data analysis with real time sampling enables our algorithms to tune the application level data transfer parameters accurately and efficiently to achieve close-to-optimal end-to-end data transfer throughput with very low overhead. Our experimental analysis over a variety of network settings shows that HARP outperforms existing solutions by up to 50% in terms of the achieved throughput.**

I. Introduction

As the trend towards data-intensive applications continues, the users have to put a great effort into efficiently moving large datasets between different sites. The effective use of the available network bandwidth and optimization of the data transfer throughput has been crucial for the end-to-end performance of most commercial and scientific applications despite the multi-gigabit optical network offerings. The majority of the users fail to obtain even a fraction of the theoretical speeds promised by the existing networks but still pay high costs 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 low-level, including design of new transport protocols [8], [9], [17], [23] as well as adapting and changing an existing transport protocol for better performance [22], [26]. At a higher level, other techniques have been developed simply by using the existing underlying protocol intact, and tuning it at the application level. One common way to address protocol tuning at the application level is through the tuning of parameters such as pipelining [10], [11], parallelism [5], [9], [14], [18], [26], [28], [34], concurrency [24], [25], [27], and buffer size [7], [15], [16], [30], [32]. These parameters can be tuned at the application level without the need for changing underlying transfer protocols and can significantly improve the end-to-end data transfer performance [4], [37].

While significant performance gain can be achieved by tuning application level protocol parameters, optimal value of these transfer parameters varies depending on the dataset (i.e. file size and the number of files), network (i.e. bandwidth, round-trip-time, and background traffic on network), and end-system characteristics (i.e file system and transfer protocol chosen). Thus, finding the best combination for these parameters is a challenging task. For instance, pipelining helps transferring multiple files back-to-back without waiting for the acknowledgement message in the control channel, but the size of the transferred files must be small to benefit from it. Pipelining may even cause throughput decrease when set to high values for large files. Instead, the large files would benefit from being divided into smaller chunks and being transferred over the network through multiple parallel streams. Similarly, both small and large files would benefit from concurrency, meaning simultaneously transferring multiple files over different transfer streams/channels. On the other hand, opening too many connections to transfer a single file (parallelism) or multiple files (concurrency) would also degrade the throughput by increasing network congestion and causing oversubscription in the file system. Optimal values for these parameters would depend on many factors described above. In our previous work, we have developed heuristic-based dynamic optimization algorithms [4] to determine the best combination of these parameters by using network and dataset characteristics (i.e bandwidth, round-trip-time, and average file size etc.).

In this paper, we present predictive end-to-end data transfer optimization algorithms based on historical data analysis and real-time background traffic probing (HARP). We use 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. Combining historical data analysis with real time sampling enables our algorithms to tune the application level data transfer parameters (i.e. parallelism, pipelining, and concurrency) accurately and efficiently to achieve close-to-optimal end-to-end data transfer throughput with very low sampling overhead. Our experimental analysis over a variety

SC16; Salt Lake City, Utah, USA; November 2016
978-1-4673-8815-3/16/$31.00 ©2016 IEEE

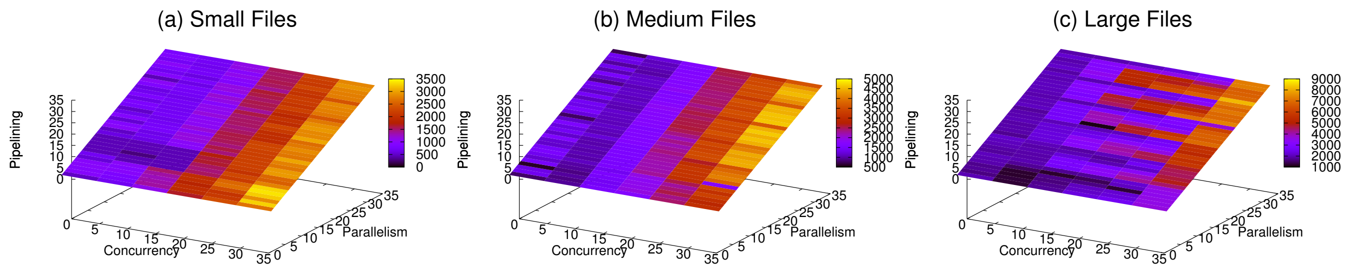

Fig. 1: Throughput variation as the parameter values change (between Stampede (TACC) and Gordon (SDSC) at XSEDE).

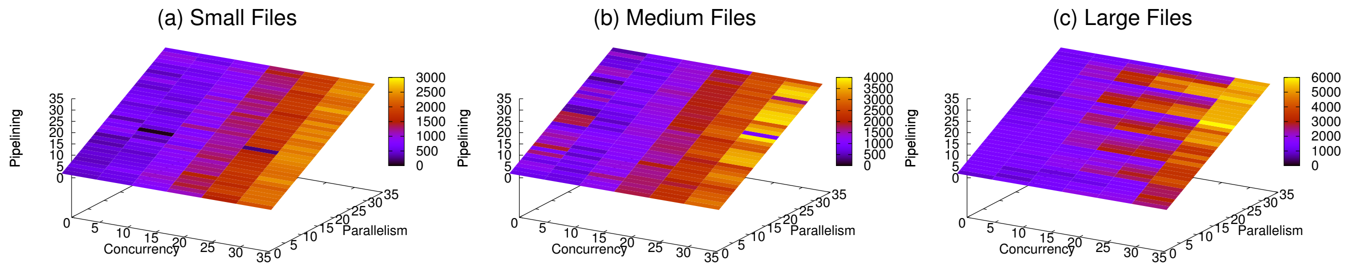

Fig. 2: Throughput variation as the parameter values change (between Stampede (TACC) and Blacklight (PSC) at XSEDE).

of network settings shows that HARP outperforms existing solutions by up to 50% in terms of achieved throughput.

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

II. Motivation

The tunable transfer parameters such as pipelining, parallelism, and concurrency play a significant role in improving the achievable transfer throughput. However, setting the optimal levels for these parameters is a challenging task and an open research problem. Poorly-tuned parameters can either cause underutilization of the available network bandwidth or overburden the network links and degrade the performance due to increased packet loss, end-system overhead, and other factors.

Among these parameters, *pipelining* targets the problem of transferring a large numbers of small files. In most control channel-based transfer protocols, an entire transfer must complete and be acknowledged before the next transfer command is sent by the client. This may cause a delay of more than one round-trip-time (RTT) between the individual transfers. With pipelining, multiple transfer commands can be queued up at the server, greatly reducing the delay between transfer completion and the receipt of the next command. *Parallelism* sends different chunks of the same file over parallel data streams (typically TCP connections), and can achieve high throughput by aggregating multiple streams and getting a larger share of the available network bandwidth. *Concurrency* refers to sending multiple files simultaneously through the network using different data channels, and is especially useful for increasing I/O concurrency in parallel disk systems.

The effect of the transfer parameters (concurrency, parallelism, pipelining) on the data transfer throughput is shown in Figures 1 and 2. The values of these parameters are changed between 1 and 32, and the colors represent the throughput obtained for a given parameter combination (the color code is given in the figure). In both Figures 1 and 2, the highest throughput is obtained in the rightmost column where concurrency level is set to 32. However, the parallelism and pipelining values for the maximum throughput differ based on the file size. Moreover, parameter values that yield the maximum throughput differ as the source and destination pair changes. For example, the maximum throughput in Figure 1 (a) is observed at (32,1,16). However, same values in Figure 2 (a) yields 12% less than the maximum throughput. Similarly, the values of maximum point in Figure 1 (a) yields 12% less throughput than the maximum in Figure 2 (a). The loss in throughput reaches up to 25% when highest throughput parameter values of the large dataset in Stampede-Blacklight pair is applied to Stampede-Gordon pair. The loss in throughput exacerbates when the optimal parameter values found in one setting are used in a totally different setting, such as the best parameter combination found in a network with enhanced distributed storage subsystem applied to a network with a single disk storage subsystem.

In addition to being dependent on network and storage subsystem specifications, the optimal parameter values are also affected by the background traffic on the network. Figure 3 shows how transfer throughput changes with different parameter value combination when network has more background traffic than Figure 2 (c). While highest throughput is achieved when values (32,8,32) are used under light background traffic, (32,16,4) achieved the best throughput under heavy network traffic. The best parameter combination for light background

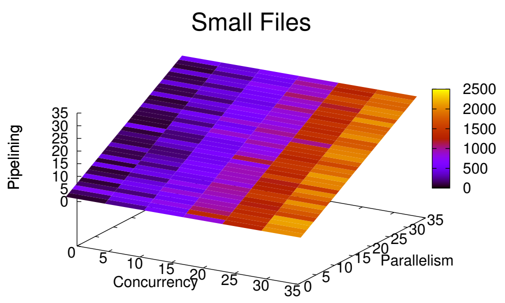

Fig. 3: Throughput variation as the parameter values changes for transfers between Stampede (TACC) and Blacklight (PSC) at XSEDE with medium background traffic.

traffic obtained 53% less throughput than the maximum observed under heavy background traffic. Hence, in order to find the best parameter combination that maximizes the achieved transfer throughput, (i) network and end-system settings (i.e., bandwidth, RTT, storage I/O throughput etc.), (ii) dataset characteristics (i.e., file size and file count), and (iii) the current system load have to be taken into consideration.

III. Overview of HARP

HARP combines three approaches of application-level data transfer tuning and optimization: *i)* heuristics; *(ii)* real-time probing; and *(iii)* historical data analysis. Heuristic algorithms [2], [4] compute transfer parameters through calculations on the dataset and network metrics. For example, the value of pipelining is calculated by dividing the bandwidth-delay-product (BDP) to the 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 [6]. However, heuristics fail to capture the dynamic changes in the network and end-system specific settings, including the real-time background traffic. Real-time probing based approaches [36], [37] find optimal values of protocol parameters by running a sequence of sample transfers (probes) using different metric values. They take a portion of the original dataset and transfer it with initial metric value (generally 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, it may bring too much probing overhead to accurately discover the optimal values of parameters. Finally, historical data methods [19], [31] model data transfer throughput based on dataset, network, and protocol parameter metrics. In order to capture change in the network load, they either rely on recent historical data [19] or run sample transfers [31]. HARP runs sample transfers similar to probing based solutions, but the number of sample transfers are way less than it is in probing based algorithms. HARP benefits from heuristic solutions to determine parameter values of sample transfers. Since poor choice of parameter values in sample transfer may affect overall throughput, taking advantage of heuristic algorithms may alleviate the sample

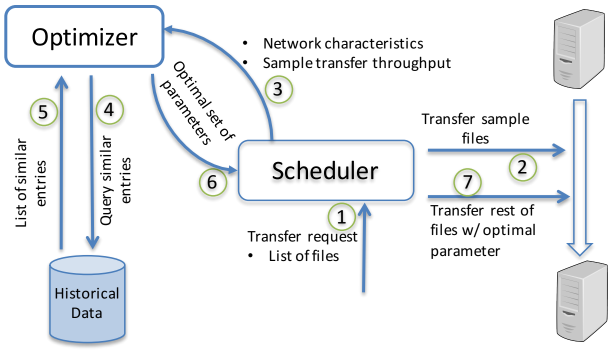

Fig. 4: Flow of operations in HARP.

overhead. Finally, HARP models transfer throughput similar to historical data based solutions. While models driven by Kettimuthu et al. [19] and Nine et al. [31] require offline analysis and can 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 extensible similarity detection algorithm.

HARP is composed of two main modules which are Scheduler and Optimizer as shown in Figure 4. When a data transfer request is submitted to the Scheduler, it first categorizes files in the transfer request into 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 (step 3), it passes this information along with the network/dataset characteristics (step 4) to Optimizer to determine the similar entries in the historical data (step 5). Then, Optimizer identifies similar entries (step 6 and 7) and runs regression analysis to derive an equation that relates transfer parameters to the transfer throughput. Then, the derived model is solved for the maximum value (i.e transfer throughput) and corresponding parameter values are obtained. After 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, it returns the parameter combinations to Scheduler to schedule the transfer of the rest of the dataset with the calculated parameter values.

A. Transfer Scheduler

HARP's Scheduler is responsible for managing data transfer executions between end points. It first divides files into groups (aka chunks) according to the file size (Tiny, Small, Medium, and Large) (line 3 of Algorithm 1). Then, it runs one sample transfer for each chunk to learn about achievable throughput of each chunk. $ST[i]$ refers to sample transfer throughput for i. In order to minimize the overhead of sample transfers, Scheduler takes advantage of heuristics [4] to determine the parameter values of the sample transfer. Although heuristic calculations may pick suboptimal values, it is mostly better than default or random values. Regarding the size of dataset used in the sample transfers, we simply used $2 * Bandwidth(MB)$ as a baseline with the additional assumption of having at least two

Algorithm 1 – Scheduler of HARP

```
1: function TRANSFER(source,destination,BW,RTT, algorithm)
2:     $allFiles = getListOfFiles()$
3:     $chunks = partitionFiles(allFiles)$
4:     for $i = 0; i < chunks.length; i++$ do
5:         $sampleFiles = chunks[i].split(SAMPLING\_SIZE)$
6:         $(cc, p, pp) = findParamsViaHeuristic(sampleFiles, BW, RTT)$
7:         $ST[i] = transferChunk(sampleFiles, cc, p, pp)$
8:     end for
9:     $maxCC = 1$
10:    $TT = 0$
11:    for $i = 0; i < chunks.length; i++$ do
12:        $cc_{est}[i], p_{est}[i], pp_{est}[i], UT[i] = runOptimizer(ST[i],$
               $BW, RTT, metadata(chunk[i]))$
13:        $TT += UT[i]$
14:        $maxCC = max(maxCC, cc_{est}[i])$
15:    end for
16:    for $i = 0; i < chunks.length; i++$ do
17:        $weight[i] = chunks[i].size * \frac{TT}{UT[i]}$
18:        $totalWeight += weight[i]$
19:    end for
20:    for $i = 0; i < chunks.length; i++$ do
21:        $cc' = max(cc_{est}[i], \lfloor maxCC * \frac{weight[i]}{totalWeight} \rfloor)$
22:        $startTransfer(chunks[i], cc', p_{est}[i], pp_{est}[i])$    ▷ Asynchronous operation
23:    end for
24: end function
```

files in the dataset to be able to try concurrency value greater than one. Using too small dataset in sample transfers might cause misleading results since dataset might fall short to reach the maximum achievable throughput. Yet, too large dataset might deteriorate overall transfer throughput by taking long time to finish if the parameter values used in sample transfer turns to be far from the optimal values. However, optimization of sampling transfer size is not in the domain of this work so HARP uses a meaningful predefined value.

After real-time probing is completed, Scheduler sends the results to Optimizer along with the dataset and network settings (line 12). Optimizer returns values for protocol parameters $(cc_{est}, p_{est}, ppq_{est})$ along with unit throughput, UT (line 12). UT refers to throughput of a chunk if it is run with concurrency value 1. It is used to determine how channels will be distributed among chunks when chunks are run simultaneously. While Optimizer finds optimal values assuming each chunk will run separately, Scheduler prefers to run multiple chunks simultaneously to benefit from multi-chunk approach as presented in our earlier work [4]. While we can use parallelism and pipelining values returned by Optimizer, concurrency must be adapted to multi-chunk transfer scheme. Even though concurrency has significant impact on throughput (especially in when parallel file systems are in use), it also causes the highest overhead at the end systems and network by creating multiple processes. Thus, it may not be possible, yet undesirable, to open as many channels (concurrency) as each chunk asks. To overcome this inconsistency, Scheduler computes the maximum concurrency ($maxCC$) of all chunks (line 14) which is then distributed among chunks based on their weights (line 21). Weight of a chunk is proportional to size of a chunk and inversely proportional to ratio to $\frac{UT}{TT}$ where TT refers to sum of all UTs (line 17). Once parameter values of chunks are determined, Scheduler runs them concurrently.

Specs	XSEDE Stampede-Gordon	DIDCLAB WS1-WS-2	EC2 I1-I2
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 Throughput (MB)	1200	90	320

TABLE I: Network specifications of the test environment.

B. Optimizer

Optimization module of HARP aims to determine the optimal parameter values for the transfer metrics for an intended data transfer with the help of historical data. Thus, it heavily depends on the quality and quantity of dataset to make the best decisions. Quality stands for how well the dataset captures variation in the network such as the background traffic on the network. Quantity is important (i) to detect outliers and (ii) to derive accurate models.

Data Collection: We collected our historical transfer data on XSEDE [35], a production-level high-speed WAN, and DIDCLAB at UB, a dedicated LAN, networks. The network and storage configurations are given in Table I. Four different file sizes are used which are Tiny (varying from 1MB to 5MB with a total of 10GB), Small (varying from 15MB to 30MB with a total of 20GB), Medium (50MB-200MB with a total of 40GB), and Large (1GB-5GB with a total of 98 GB). Example historical data entries are given in Table II in which we kept dataset and network configurations fixed and changed protocol metric values, one at a time.

To see the effect of protocol metrics under different network loads, we tested same parameter combinations under different network loads; light, medium, and heavy background traffic. Although we can control background traffic in LAN experiments, we cannot know the exact background traffic in a shared production network, such as XSEDE. However, we observed higher and more stable throughput values when we run transfers in the night hours. Thus, data entries are collected in the night hours of the day to minimize the effect of external load. Moreover, we repeated each entry at least five times at different dates so that any outliers can be detected and neglected easily. For the medium and heavy background traffic cases, we synthetically created background traffic by running multiple memory-to-memory transfers in the background during the data collection. Over a 12-week period, we collected statistics on 21K data transfers. Since we only kept metadata (number of files, average file size, date etc.) of data transfers, the amount of storage to store historical data was around 4MB. Even though we did not experience storage limitation in our experiments, one can put a time limit to remove old entries and keep historical data size at a certain level.

1) Data Filtering and Grouping: When Optimizer receives a request from Scheduler which consists of a dataset, network characteristics and sample transfer throughputs, the first operation it does is to filter similar entries from the data store where historical data is kept. Since it is possible that the

Entry Number	Source	Destination	Bandwidth (Gbps)	RTT (ms)	Avg File Size (B)	Number of files	Parallelism	Concurrency	Pipelining	Throughput (Gbps)
1	Stampede	Gordon	10	40	3150088	3409	1	1	1	491.0
2	Stampede	Gordon	10	40	3150088	3409	1	1	2	958.1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
216	Stampede	Gordon	10	40	3150088	3409	32	32	32	2827.6
217	Dicdlab-ws1	Dicdlab-ws2	1	0.2	652486	3293	1	1	1	386.8
218	Dicdlab-ws1	Dicdlab-ws2	1	0.2	652486	3293	32	16	32	454.2
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
432	Dicdlab-ws1	Dicdlab-ws2	1	0.2	652486	3293	1	1	1	640.9

TABLE II: Example of historical data entries.

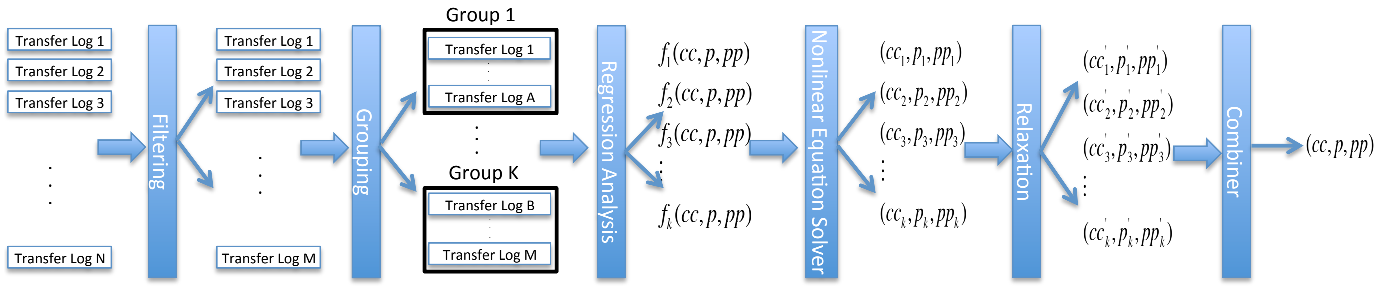

Fig. 5: Flow of operations in HARP's Optimizer.

data store may not have exactly matching entries for a given dataset/network characteristics, Optimizer uses a weighted cosine-similarity function (shown in Equation 1) to measure the similarity of historical data entries to the intended transfer.

$$cos(\theta) = \frac{\sum_{i=1}^{n} \|A_i\| \|B_i\|}{\sqrt{\sum_{i=1}^{n} \|A_i^2\|}\sqrt{\sum_{i=1}^{n} \|B_i^2\|}} \tag{1}$$

Cosine-similarity uses set of features and calculates the degree of alikeness based of how the features of objects are close to each other. In Equation 1, A and B refers to values for feature set of two instances. In the context of data transfer, feature set consists of dataset and network settings of transfers e.g. bandwidth, round-trip-time, $\frac{bandwidth-delay-product}{buffer-size}$, chunk type (Tiny, Small, etc.), file size, and file count. $\frac{bandwidth-delay-product}{buffersize}$ is used to determine if use of parallel streams would help to overcome TCP buffer size limitation. Even though some of the features are related to each other, e.g. chunk type and file size, we wanted to be as much specific as possible in terms of similarity detection. For example, if BDP is 40 MB, 1MB and 1KB sized files will be put into Tiny chunk and 1GB will be in Large chunk according to our dataset partitioning method. However, if we just use file size to compare similarities, 1MB file size will have same similarity value when compared to 1KB and 1GB files. To address such misclassification, we evaluate some features in multiple ways to have more accurate similarity detection. Moreover, we normalize feature vectors of historical data entries to try to keep ranges of properties as close as possible otherwise, one property may overweight the similarity value if value of a property is larger than others. Finally, since each feature has different impact on file transfer throughput, we assigned weights to them based on our initial effort to apply regression to whole historical data. Although the accuracy of the regression is low, it gives a clue about the weight of each property on achieved transfer throughput. Hence, we used (2,2,10,10,3,1) weight values for the feature set Bandwidth, RTT, $\frac{BDP}{buffersize}$, chunk type, file size and file count), respectively. Since there are other factors aside from dataset and network characteristics that affect transfer throughput (e.g background traffic, disk I/O performance etc.), Optimizer runs additional step of categorization using the results of sample transfers which is explained in Section III-B3.

Once Optimizer calculates similarity value (according to the intended transfer) for each entry in the historical data, it picks entries with similarity value larger than the threshold. We initialized the threshold value to 0.99 and decreased it until we have at least 3K entries. Since cosine-similarity does not consider background traffic, the selected entries will have transfers with different background traffic. To overcome hidden variable problem, we grouped set of entries that are not only same in regard to dataset and network characteristics but also collected at approximate times. For example, first 216 entries in Table II will be grouped as a one set of experiments because they are all run in around same time (which can be inferred by time of transfer) and have same values for dataset and network settings. Although it is possible that while any of 216 entries are running, background traffic may change drastically, those will be identified and ignored in the modeling phase. During data collection, we ran each dataset with all possible combination of parameter values (from (1,1,1) to (32,32,32)). Thus, when we group historical data entries based on data collection time, each group will have 216 entries. Thus, Optimizer will have at least 15 groups (set of entries) (3K/216) at the end of similarity detection phase.

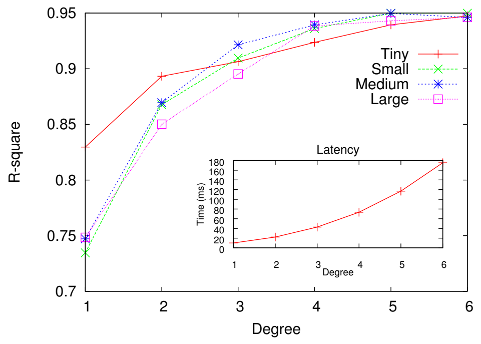

Fig. 6: Comparison of multiple degrees of polynomial regression

2) Regression Analysis and Nonlinear Equation Solver: After similar entries are filtered, a model is driven for each group of entry using polynomial regression that relates transfer protocol parameters to transfer throughput as in Equation 2. Thr_i refers to the equation derived for i^th historical data group where $1 < i < N$ and N is the total number of historical data groups after filtering phase. Since entries in a group shares same network and dataset characteristics but differ in values of protocol parameters, we can derive a model on this data that relates protocol parameters to transfer throughput as shown in Equation 2. By grouping entries with similar network and dataset metrics, we can decrease the number of input parameters to the model which leads to higher success in fitting regression. We compared several degrees of polynomial regression in terms of accuracy and speed in Figure 6. According to the results, cubic polynomial regression appears to be well suited as it yields R^2 values around 0.9 for all file types and can be computed relatively faster. Although hexic regression can increase R^2 to 0.95, time to derive it is more than four of cubic regression's. Thus, we have derived cubic polynomial equations to relate protocol parameters to transfer throughput. Since historical data may contain outliers, we discarded groups whose derived regression has R^2 value smaller than 0.6.

$$T_i = f_i(cc, p, pp) \tag{2}$$

After cubic polynomial equations are derived for each group of historical data, $f_1, f_2, ..., f_k$, Optimizer evaluates them for the values used in sample transfers to find the estimated throughputs, T_1,T_2,...,T_k as shown in Equation 3. ϵ_i refers to the difference between throughputs estimated by f_i and obtained by the actual sample transfer, Thr_{act}. In order to prioritize historical data entries that are exposed to similar background traffic compared to current traffic, Optimizer assigns weights to the equations based on their accuracy in estimating throughput of sample transfers. In order to assign weights to equations, we classify them into groups using density based clustering technique, DBScan based on ϵ values. Each equation in a class is assigned a same weight (w) and the weight of a class is calculated as $2^0, 2^1, ..., 2^{k-1}$ where classes are sorted in descending order based on ϵ values. Weights play significant role in (i) distinguishing historical data collected under different background traffics and (ii) compensating similarity based filtering for possible misclassification due to unacknowledged factors of data transfers such as background traffic. We have observed that DBScan generally return 4-6 groups.

$$\epsilon_i = T_{act} - f_i(cc_0, p_0, pp_0) \tag{3}$$

After weights of equations are found, Optimizer finds parameter combination for each equation that returns the highest throughput using the non-linear programming solver (fmincon of Matlab). It scans solution space for the variables, (cc_i, p_i, pp_i), that returns maximum throughput, $Tmax_i$, as shown in Equation 4. Optimizer combines the values found by each equation by taking weighted average as shown in Equation 5.

3) Variable Relaxation and Combiner: Once corresponding parameter values for maximum throughput are found for each f_i. $1 < i < k$, Optimizer runs relaxation process during which the values of parameters are lowered if change in the estimated throughput stays within a reasonable range. For example, pipelining has very little contribution, if not none, to the throughput on large files, but the optimal point may estimate large pipelining value for a marginal gain. Optimizing cost of energy consumption can be shown as another reason for the importance of relaxation process as it is shown that [1], marginal increase in transfer throughput by means of using large values of parameter values may lead to considerable increase in power consumption.

In the relaxation phase, Optimizer evaluates smaller values for each parameter until the new estimated throughput is larger than certain percentage of the original estimated throughput. Assume (32, 20,24) is calculated as optimal values for concurrency, parallelism and pipelining for an equation f and $(32, 20, 24)$ returns $Tmax$ as a throughput. The relaxation process will evaluate smaller values for concurrency starting from 31, while keeping parallelism and pipelining same, until $f(cc', 20, 32)$ returns $Tmax^{'}$ where $Tmax' < \rho * Tmax$. We have used 0.9 for values for ρ as it appeared to be sufficient to successfully identify marginal contributions parameters. Once cc' is determined, T_{max} is updated to T'_{max} and relaxation process is then applied to parallelism and pipelining as well. Since we apply relaxation for each parameters one by one, final $Tmax^{'}$ would be $0.72 * Tmax$.

$$Tmax_i = f_i(cc_i, p_i, pp_i) \tag{4}$$

$$cc_{avg} = \sum_{i=1}^{N} \frac{cc_i * w_i}{w_{total}} \qquad p_{avg} = \sum_{i=1}^{N} \frac{p_i * w_i}{w_{total}} \qquad pp_{avg} = \sum_{i=1}^{N} \frac{pp_i * w_i}{w_{total}} \tag{5}$$

Cost Analysis of HARP: HARP runs sample transfers and applies data modeling on the fly so comes with an overhead. To minimize the overhead, we pipelined the transfer sampling process with optimization process at the best effort. Instead of waiting for each chunk's sample transfer to be completed,

T_H **Gain** (%)	T_S **Slowdown** (%)	D_S ($\times BW$)	**Min. Chunk** Size ($\times BW$)
10	80	2	43
10	50	2	30
10	30	2	12.8
30	50	2	13.3
30	30	2	7.1
50	50	2	10
50	30	2	5.7

TABLE III: Cost analysis of HARP under different scenarios

we run Optimizer for a chunk as soon as its sample transfer is completed so that Scheduler and Optimizer can operate simultaneously. For example, once Scheduler finishes sample transfer for small chunk, it starts running sample transfer for medium chunk. While sample transfer for medium chunk runs, it passes sample transfer throughput of small chunk to Optimizer so that it runs calculations and returns values for protocol parameters. Since Optimizer can finish an estimation calculations in around two seconds for each chunk, 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 chunk.

$$t_0 = \frac{D}{Thr_0} \tag{6}$$

$$t_H = \frac{D - D_S}{Thr_H} + \frac{D_S}{Thr_S} + c \tag{7}$$

Equation 6 shows the duration of the data transfer when HARP is not used. D refers to data size and Thr_0 refers to the obtained throughput. When HARP is used, the duration of the data transfer is determined by Equation 7 in which D_S refers to the data size of the sample transfer and Thr_H and Thr_S refers to the throughputs obtained in sample transfer and actual dataset transfers, respectively. c refers to the cost of Optimizer for running the optimization process for the last chunk.

In Table III, we analyzed the cost of HARP for different values of Thr_H, Thr_S and D_S. We calculated the minimum chunk size for HARP to amortize the cost it induces. T_H *Gain* column represents $\frac{Thr_H - Thr_0}{Thr_0}$ which stands for throughput gain when estimated parameter values are used. Although the gain will be much higher when HARP is compared with Globus Online [2] and PCP [36], we compared HARP against heuristics we porposed in our earlier work [4] which outperform Globus Online and PCP by a significant margin. T_S *Slowdown* column represents the ratio of transfer throughput decrease when sample transfers run. For example 80% slowdown means $Thr_S = (1 - 0.8) * Thr_0$. D_S and *Min Chunk Size* columns are represented in bandwidth type which refers to bandwidth of the network in bytes. For 10 Gbps network, D_S equals to 2.5 GB ($2 * \frac{10}{8}$).

In the worst case scenario, the gain of HARP is 10% and the sample transfers are 80% slower than Thr_0, minimum chunk size that HARP amortizes the cost is $43 * BW$. The minimum chunk size to benefit from HARP reduces as the the throughput gain increases or sample transfer slowdowns reduces. Our observations on the tests we ran in XSEDE, AWS and DIDCLAB networks, we observed that slowdown mostly stays lower that 50% and the gain ranges from 10% to 80%. Hence, the results we present in Section IV show that HARP mostly outperforms the heuristic algorithms under light traffic and improves the overall throughput significantly under medium and heavy background traffic cases.

IV. Experimental Analysis

We compared HARP against heuristic (Globus Online [2], Single Chunk, and ProActive Multi-Chunk [4]), probing based [36], and hysteresis based [31] algorithms. Globus Online (GO) separates dataset into chunks based on file size and uses predefined values for protocol parameters for each chunk's transfer. Single Chunks (SC), again, separates dataset based file size and transfers them one by one with the protocol parameter values found by simple arithmetic calculations using dataset and network metrics. ProActive Multi-Chunk (ProMC) creates chunks and determines values of protocol parameters similar to SC, but instead of transferring each chunk by itself, it runs multiple chunks at the same time in order to minimize the effect of small files on overall transfer throughput. Since SC and ProMC require user input for upper bound of concurrency level of a transfer, we have set it to 10 as they seem to be performing the best when maximum concurrency is set to 10 [4]. PCP [36] employs a divide-and-transfer approach similar to SC and GO. It determines protocol parameters by running several sample transfers. Finally, ANN+TO models transfer throughput based on historical data and runs sample transfer to learn the current load on the network. Similar to SC and Globus Online, it transfers chunks one by one due to which its overall performance for dataset with mixed data files is dominated by the throughput of small files.

We tested HARP both at the networks for which historical data have and have not matching entries in terms of source-destination pairs of data transfers. Datasets used in the experiments are different from the ones used in historical data collection process. While datasets in data collection process are homogeneous (e.g. 10000 of 1 MB files are used for small file types), experiment datasets are generated such that while all file types (small, large etc.) exist, file sizes in a file type are determined randomly. Our experiments on XSEDE (from Stampede to Gordon) and DIDCLAB networks are the ones historical data have matching entries and another XSEDE (from Gordon to Stampede) and AWS experiments are the ones historical data does not have entries with same network settings . We also included *Maximum* throughput in the results which refers to the highest observed throughput for a given network and dataset either in experiments and historical data. All the experiments are run at least five times.

We first experimented with transfer of datasets that only have one type of file size; either all large or small files. Figure 7 shows the comparison of GO, ANN+OT, SC, ProMC, and HARP. Size of datasets are 45 GB and 92 GB for small and large files, respectively. While SC and ProMC achieve similar throughput for small file transfer, they differ in large file transfer. This is because of the way SC calculates concurrency

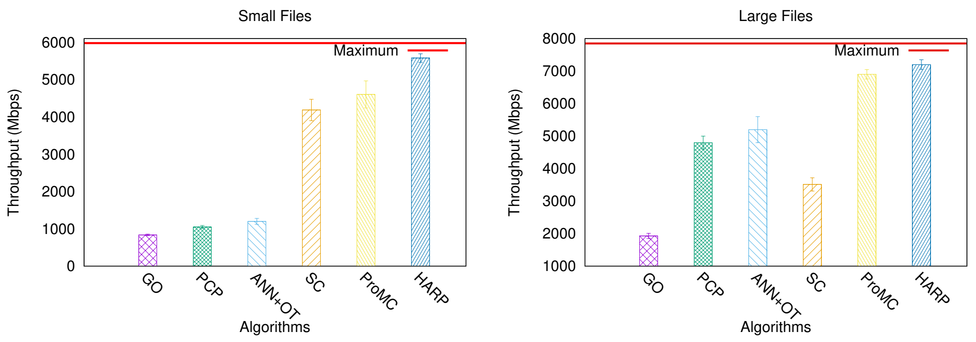

Fig. 7: Single type file transfers between Stampede (TACC) and Gordon (SDSC) on XSEDE.

level of a chunk. While ProMC uses all available channels (in this case it is 10), SC may prefer using less number of channels than available. SC determines the number of channels by taking minimum of what it calculates and what is given by user as upper bound. SC calculates more than 10 channels for small files so, it uses concurrency value 10 as we set upper bound to 10. On the other hand, it estimates concurrency value 2 for large files thus, yields lower throughput than ProMC. While ANN+OT performs worse than SC in small file transfers, it outperforms SC by 32% in large file transfers since it predicts concurrency value larger than what SC calculates. PCP also performs worse than SC for small files which is due to overwhelming effect of sample transfers. Even though it also runs sample transfers for large files, small files are more sensitive to values of concurrency and pipelining than large files. When small values of concurrency and pipelining are used during probing process, it takes long time to finish transfer which affects overall transfer considerably. Moreover, HARP outperforms SC and ProMC by around 25% for small files. However, HARP outperforms ProMC only around 5% for large files. Digging into details, we found out that Optimizer estimates concurrency level in 10-13 range for large files which is close to what ProMC is set to run in this experiment. Although HARP yields higher throughput after Optimizer estimates concurrency, due to overhead of sampling and optimization process, its becomes marginal. 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 some degree of knowledge on transfer parameters as well as dataset characteristics. Finally, HARP outperforms ANN+OT for both small and large files which proves that HARP does better job in modelling and finding best values of optimal parameters.

Figures 8 and 9 are the results that we obtained in the networks that historical dataset contains exact matching entries. We have tested algorithms under three different network loads (light, medium, and high background traffic as described in "Data Collection" section). In both networks, the performance of most algorithms decreased by 50-300% as network load increases.

Algorithms that transfers one chunk at a time (GO, ANN+OT, PCP, and SC) exhibit poor performance in XSEDE because their overall performance is pulled down by the throughput of small file transfers. While the maximum transfer throughput is observed as high as 8 Gbps,GO, ANN+TO, PCP, and SC achieve less than 3 Gbps under light background traffic. On the other side, multi-chunk algorithms (ProMC, and HARP) are able to deliver $\sim$7Gbps. As discussed in Section III-B3, HARP-requires data size to be greater than certain amount to outperform the heuristics. The size of dataset used in Figure 8 was 130GB which is only enough to cover the overhead imposed by HARP. When dataset size is increased to 260GB, throughout of HARP reached to 8 Gbps as HARP's overhead is alleviated by the increase in overall transfer duration. Unlike light background traffic case, HARP gains 36% and 48% more throughput than ProMC even for smaller dataset size (130GB) under medium and heavy background traffic cases by taking advantage of historical transfer information. The reason why ProMc performs worse as the network load varies is its traffic-agnostic parameter estimation approach. One may claim that ProMC can learn about network load by running sample transfers. However, without having historical information, it cannot interpret probing results. A simple way to interpret probing is done by PCP algorithm which runs several probings and increments parameter values until probing throughput decreases. However, results show that it fails to capture high overall transfer throughput as the number of probings becomes high to accurately identify "right value" for parameters. The throughput of SC algorithm is dropped drastically from 3.2 Gbps to 750 Mbps as network load increases. Similarly, throughput of GO suffers significantly as network traffic increases. ANN+OT and PCP are able to adapt protocol parameters accordingly and outperform SC and GO under high network loads. However, overhead of sampling and "one chunk at a time" policy limits their overall performance to less than 1 Gbps in heavier network loads.

We picked two DIDCLAB servers (WS-1 and WS-2) to test HARP in a LAN with single disk storage subsystem. Figure 9 presents the performance comparison of algorithms. When background traffic is low (Figure 9 (a)), SC algorithm performs better than ProMC since opening multiple processes to handle disk I/O operations deteriorates the disk I/O performance while not improving network throughput. However, as background

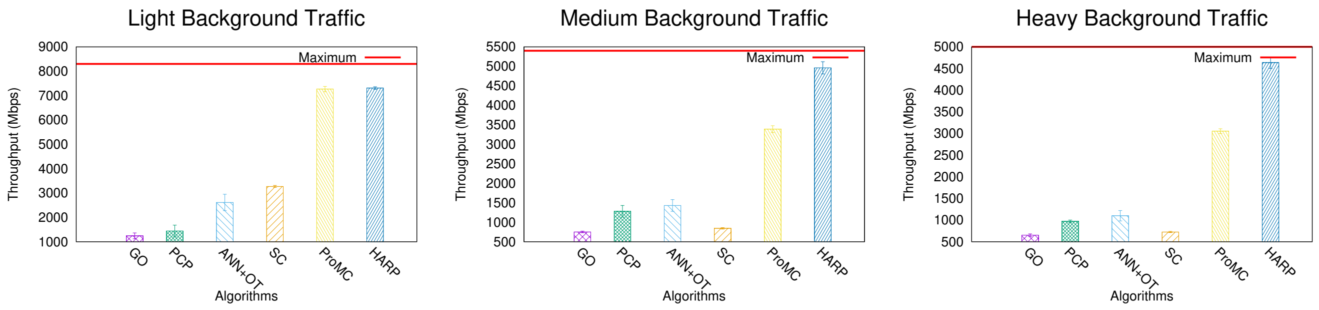

Fig. 8: Comparison of algorithms for data transfers between Stampede (TACC) and Gordon (SDSC) on XSEDE.

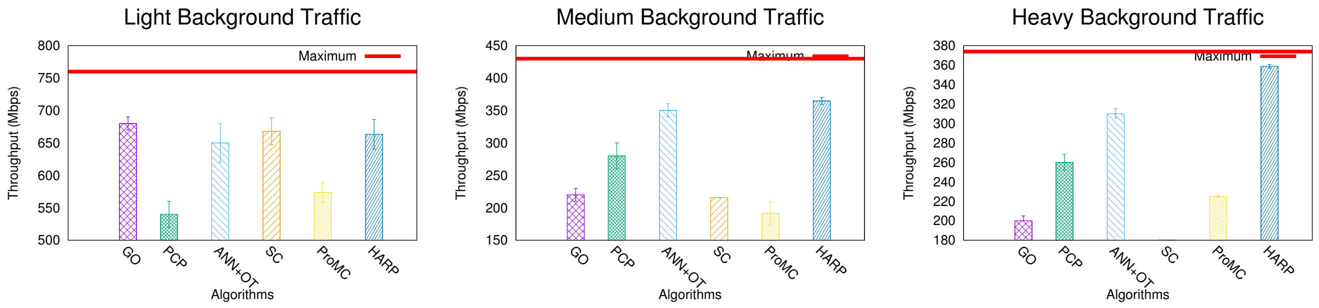

Fig. 9: Comparison of algorithms for data transfers between WS-1 and WS-2 on DIDCLAB.

traffic increases (Figure 9 (b) and (c)), multi chunk transfer helps obtaining higher network throughput. In Figure 9 (a), we can see that HARP-yields 650 Mbps throughput which is bounded by disk I/O throughput. On the contrary, network throughput becomes the bottleneck when network is highly congested. Hence, HARP achieves bigger end to end transfer throughput by adapting protocol parameters to varying network conditions.

Since heuristic algorithms are unaware of disk subsystems when calculating protocol parameters, they pick parameter values solely based on network and end system configurations. However, this simple heuristic approach might be misleading when transfer throughput is limited by disk I/O performance and disk I/O throughput decreases as the number of active threads increases. Hence, HARP can outperform heuristic algorithms even when relatively small dataset with the help of historical data. When there is no background traffic (Figure 9 (a)), HARP achieves 13% higher throughput than ProMC. As the network load increases, HARP achieves 47% and 37% more of ProMC's throughput for medium and high network load experiments. ANN+OT outperforms heuristics under all network loads but falls short to compete with HARP-since uses single chunk approach.

In order to test the effectiveness of HARP for networks that have no matching entries in historical data, we have run experiments on XSEDE and Amazon EC2. Although same pair of servers are used in XSEDE experiments, we have transferred dataset in a reverse path of historical data entries. Namely, historical data has the logs for transfer 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 to be identical of Stampede-Gordon transfers, the results shows us that the maximum achievable throughput is different than Stampede-Gordon transfers basically due to (i) end system storage speeds are different for read and write operations, (ii) at any given time free network bandwidth is less than what is observed in reverse direction. Hence, Gordon-Stampede is a good example to see how HARP performs on the networks that have a similar but not exact entries in historical data.

As opposed to Stampede-Gordon transfers, Gordon-Stampede transfers are more disk I/O bound which can be deduced by looking at throughput change as the background traffic increases. While throughput decrease ranges in 50-300% in Stampede-Gordon transfers as network load increases, it stayed around 10-100% in Gordon-Stampede transfers. HARP, ANN+OT, and PCP are affected the least by increased network traffic compared to heuristics since they probe network status at the beginning of transfer and picks parameter values accordingly. For example, HARP gained 13% more throughput than ProMC under light background traffic. The improvement ratio increased to 24% as throughput of ProMC dropped by 24% under heavy network throughput while throughput of HARP only dropped by 11%. Moreover, the difference between maximum observed throughput and throughput obtained by HARP increased when compared to earlier experiments. This is an expected behavior since Optimizer selects logs of Stampede-Gordon transfers during modeling phase and optimal values for protocol metrics for Gordon-Stampede transfers are not the best ones for Stampede-Gordon transfers as explained in Section II.

Finally, we tested HARP in Amazon EC2 for which we

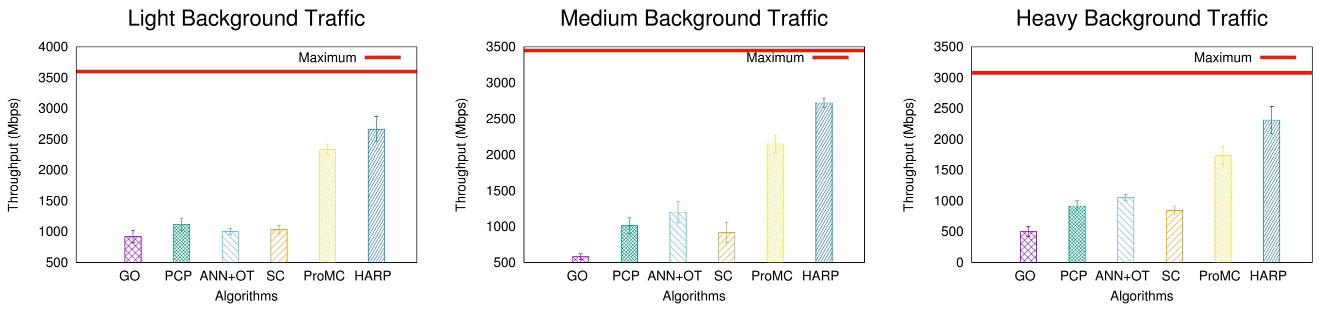

Fig. 10: Comparison of algorithms for data transfers between Gordon (SDSC) and Stampede (TACC) on XSEDE.

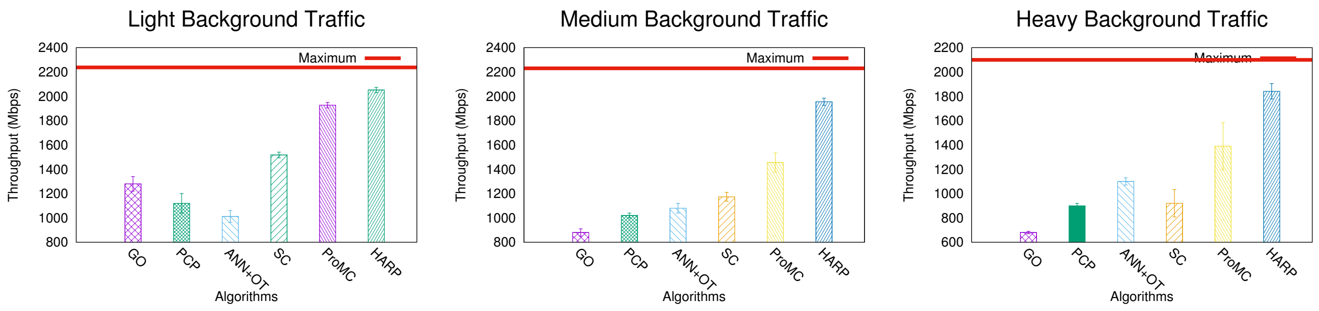

Fig. 11: Comparison of algorithms for data transfers between two c3.8xlarge instances on AWS EC2.

used two c3.8xlarge instances with network specification given in Table I. We have used Provisioned IOPS EBS storage volume as it offers the highest disk I/O throughput (around 320 MB/s). However, we were able to achieve around 265 MB/s disk throughput because of file operation overheads as a result of having too many small files. Similar to XSEDE experiments, algorithms that transfer multiple chunks simultaneously (ProMC and HARP) performed better than single chunk algorithms at all background traffic loads as shown in Figure 11. Similar to the Gordon-Stampede experiments, transfer throughputs are not as much affected as Stampede-Gordon experiments by increased network load since transfer throughputs are again limited by disk I/O throughput.

Due to similarities in network settings, cosine-similarity favors WAN transfer logs over LAN transfer logs when filtering similar entries in historical data. Since EC2 network resembles to XSEDE network in context of yielding higher disk I/O throughput as the number of concurrent transfer increases and having a smaller buffer size than BDP, XSEDE entries-based derived model works well in EC2. Hence, HARP outperforms ProMC by 5% in no background traffic case. The difference reaches to 34% and 32% under medium and high network loads as ProMC fails to adapt protocol parameters to varying system load.

A. Accuracy of the Model

Table IV shows values of the parameters for transfers in Wide Area and Local Area networks under different background traffic cases. Since transfer parameters have different impacts on different file sizes, we have gathered transfer parameters for each file type separately.

Optimizer is able to differentiate WAN and LAN transfers by picking high concurrency values for WAN transfers and low concurrency values for LAN transfers. For different file size in WAN and LAN transfers, it calculates high concurrency values for small file types and high parallelism values for large file types which seem to be the case also in Figures 1, 2, and 3. Although the optimal values for concurrency and parallelism might be determined as 32 in Nonlinear Equation Solver, Relaxation process decreases them a bit to avoid overloading network and end systems. In addition, it mostly picks higher parallelism values as network becomes more congested in order to receive higher share in network resources.

After *Filtering* process of Optimizer selects similar entries from historical data and *Grouping* categorizes them, we allot 30% of all groups as test data and use the rest to train the model. To measure correctness of the derived model, we first calculate optimal parameter values using training data. Let's say it returns $(cc_{training}, p_{training}, pp_{training})$ for corresponding throughput $Thr_{training}$. Then, we use test data and apply polynomial regression to derive model, f_{test}, and find optimal parameter values, $(cc_{test}, p_{test}, pp_{test})$, and estimated throughput, Thr_{test}. Then, instead of directly comparing throughputs $Thr_{training}$ and Thr_{test}, we calculated throughput $Thr_{projected} = f_test(cc_{training}, p_{training}, pp_{training})$ in order to project how close parameters of training data are to the optimal parameters of test data. Direct throughput comparison may not be accurate because test data and training data might have been exposed to different background traffic, thus maximum throughput of two sets of data might be different even if optimal parameters are same. So, to measure correctness of regression analysis, we calculated validation

	Stampede-Gordon (WAN)								WS1-WS2 (LAN)							
File Type	Tiny		Small		Medium		Large		Tiny		Small		Medium		Large	
Traffic	Light	Medium	Light	Medium	Light	Medium	Light	Medium	Light	Medium	Light	Medium	Light	Medium	Light	Medium
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	5.5	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 IV: Sample transfer metrics and accuracy values from HARP's Optimizer

accuracy as $\frac{|Thr_{test} - Thr_{projected}|}{Thr_{projected}}$. We also listed estimation accuracy which measures closeness of estimated throughput to the actual 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 optimal parameters stays same.

Validation accuracy of HARP is always above 85% which indicates success of regression analysis in modelling transfer throughput. While accuracy of throughput estimation in LAN is more stable and comparatively high, it is worse in 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 file type in WAN experiment, we have discovered that while maximum throughput of Tiny files in historical data never reaches beyond 4 Gbps, we have observed 5.5 Gbps in test experiments. Thus, the accuracy of throughput estimation highly depends on consistency of historical data with current network status. This can easily be handled by logging every real time transfer so that historical data can hold up-to-date information.

V. Related Work

Liu et al. [27] developed a tool which optimizes multi-file 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 [2] 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 [20], [33], 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 [3], [12], [29], [37], pipelining [6], [10], [11], and concurrent transfers [24], [25], [27]. 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. [38], Yin et al. [39], and Kim et al. [21] 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 [13], [29].

Later, Yildirim et al. presented the PCP algorithm to dynamically tune parameter values of data transfer [36]. PCP categorizes files in dataset 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. Series of sample transfers are run to determine so-called optimal value of a parameter. Although PCP does not require historical data to operate and it can adapt itself to varying network conditions, too many sample transfers are required to determine "optimal" value. Even though original dataset is used during sample transfers, overall transfer throughput are affected by sample transfer throughputs a lot as shown in Evaluation section.

In our earlier work we have proposed heuristic algorithms [4] to determine the best parameter combination by using network and dataset characteristics (i.e bandwidth, round-trip-time, and average file size etc.). Nine et al. developed ANN+OT [31] which uses historical data to derive model that relates transfer metrics to transfer throughput. It then runs real time probing in order to capture current network status.

VI. Conclusions and Future Work

In this paper, we presented predictive end-to-end data transfer optimization algorithms 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, which either cause too much sampling overhead or fail to accurately predict the correct transfer parameters. Combining historical data analysis with real time sampling enables HARP to tune the application level 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 analysis over a variety of network settings shows that HARP outperforms existing solutions by up to 50% in terms of the achieved throughput.

As a future work, we plan to form an active feedback loop for the throughput modeling to offer a dynamically evolving model based on current network conditions. This will let us detect and fix incorrect metric estimations if the Optimizer fails to find the correct entries in the first round of modeling. It will also facilitate adaptation to the changing system dynamics such as instantaneous changes in the background traffic while the transfer is running. We also plan to investigate the optimal sampling frequency and the sampling size. Although HARP already tries to minimize the probing overhead, the size and frequency of sampling can still affect the overall transfer throughput as discussed in Cost Analysis section.

References

[1] Alan, I., Arslan, E., and Kosar, T. Energy-aware data transfer algorithms. In *Proceedings of Supercomputing* (2015).

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

[3] Altman, E., and Barman, D. Parallel tcp sockets: Simple model, throughput and validation. In *Proceedings of IEEE INFOCOM* (2006).

[4] Arslan, E., Ross, B., and Kosar, T. Dynamic protocol tuning algorithms for high performance data transfers. In *Proceedings of the 19th International Conference on Parallel Processing* (Berlin, Heidelberg, 2013), Euro-Par'13, Springer-Verlag, pp. 725–736.

[5] Balakrishman, H., Padmanabhan, V. N., Seshan, S., Stemm, M., and Katz, R. H. Tcp behavior of a busy internet server: Analysis and improvements. In *Proceedings of INFOCOM '98* (March 1998), IEEE, pp. 252–262.

[6] Bresnahan, J., Link, M., Kettimuthu, R., Fraser, D., and Foster, I. Gridftp pipelining. In *Proceedings of TeraGrid* (2007).

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

[8] Crowcroft, J., and Oechslin, P. Differentiated end-to-end internet services using a weighted proportional fair sharing tcp. *SIGCOMM Comput. Commun. Rev. 28*, 3 (July 1998).

[9] Eggert, L., Heidemann, J., and Touch, J. Effects of ensemble-tcp. *SIGCOMM Comput. Commun. Rev. 30*, 1 (Jan. 2000).

[10] Farkas, K., Huang, P., Krishnamurthy, B., Zhang, Y., and Padhye, J. Impact of tcp variants on http performance. *Proceedings of High Speed Networking 2* (2002).

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

[12] Hacker, T. J., Noble, B. D., and Athey, B. D. Adaptive data block scheduling for parallel tcp streams. In *Proceedings of HPDC* (2005).

[13] Hacker, T. J., Noble, B. D., and Atley, B. D. The end-to-end performance effects of parallel tcp sockets on a lossy wide area network. In *Proc. of IPDPS* (2002).

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

[15] Hasegawa, G., Terai, T., Okamoto, T., and M, M. Scalable socket buffer tuning for high-performance web servers. In *International Conference on Network Protocols(ICNP01)* (2001), p. 281.

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

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

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

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

[20] Khanna, G., Catalyurek, U., Kurc, T., Kettimuthu, R., Sadayappan, P., Foster, I., and Saltz, J. Using overlays for efficient data transfer over shared wide-area networks. In *Proceedings of SC* (Piscataway, NJ, USA, 2008).

[21] Kim, J., Yildirim, E., and Kosar, T. A highly-accurate and low-overhead prediction model for transfer throughput optimization. *Cluster Computing 18*, 1 (2015), 41–59.

[22] Kola, G., Kosar, T., and Livny, M. Run-time adaptation of grid data-placement jobs. *Scalable Computing: Practice and Experience 6*, 3 (September 2005), 33–43.

[23] Kola, G., and Vernon, M. K. Target bandwidth sharing using endhost measures. *Perform. Eval. 64*, 9-12 (Oct. 2007).

[24] Kosar, T., and Balman, M. A new paradigm: Data-aware scheduling in grid computing. *Future Generation Computing Systems 25*, 4 (2009), 406–413.

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

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

[27] Liu, W., Tieman, B., Kettimuthu, R., and Foster, I. A data transfer framework for large-scale science experiments. In *Proceedings of DIDC Workshop* (2010).

[28] Lu, D., Qiao, Y., and Dinda, P. A. Characterizing and predicting tcp throughput on the wide area network. In *Proceedings of ICDCS '05* (June 2005), IEEE, pp. 414–424.

[29] Lu, D., Qiao, Y., Dinda, P. A., and Bustamante, F. E. Modeling and taming parallel tcp on the wide area network. In *Proceedings of IPDPS* (2005).

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

[31] Nine, M. S. Q. Z., Guner, K., and Kosar, T. Hysteresis-based optimization of data transfer throughput. In *Proceedings of the Fifth International Workshop on Network-Aware Data Management* (New York, NY, USA, 2015), NDM '15, ACM, pp. 5:1–5:9.

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

[33] Raiciu, C., Pluntke, C., Barre, S., Greenhalgh, A., Wischik, D., and Handley, M. Data center networking with multipath tcp. In *Proceedings of Hotnets-IX* (2010).

[34] Sivakumar, H., Bailey, S., and Grossman, R. L. Psockets: The case for application-level network striping fpr data intensive applications using high speed wide area networks. In *Proceedings of SC'00 ACM/IEEE conference on Supercomputing* (September 2001), ACM/IEEE, pp. 37–es.

[35] XSEDE. Extreme Science and Engineering Discovery Environment. http://www.xsede.org/.

[36] Yildirim, E., Arslan, E., Kim, J., and Kosar, T. Application-level optimization of big data transfers through pipelining, parallelism and concurrency. *Cloud Computing, IEEE Transactions on PP*, 99 (2015), 1–1.

[37] Yildirim, E., Yin, D., and Kosar, T. Balancing tcp buffer vs parallel streams in application level throughput optimization. In *Proceedings of DADC Workshop* (2009).

[38] Yildirim, E., Yin, D., and Kosar, T. Prediction of optimal parallelism level in wide area data transfers. *IEEE TPDS 22(12)* (2011).

[39] Yin, D., Yildirim, E., and Kosar, T. A data throughput prediction and optimization service for widely distributed many-task computing. *IEEE TPDS 22(6)* (2011).