Building a Wide-Area File Transfer Performance Predictor: An Empirical Study

Zhengchun Liu[2,1], Rajkumar Kettimuthu[1,2], Prasanna Balaprakash[1], Nageswara S.V. Rao[3], and Ian Foster[1,2]

[1] Argonne National Laboratory, 9700 Cass Ave., Lemont, IL 60439, USA
[2] University of Chicago, Chicago, IL 60637, USA
[3] Oak Ridge National Laboratory, Oak Ridge, TN 37830, USA
{zhengchun.liu, kettimut, pbalapra, foster}@anl.gov, raons@ornl.gov

Abstract. Wide-area data transfer is central to geographically distributed scientific workflows. Faster delivery of data is important for these workflows. Predictability is equally (or even more) important. With the goal of providing a reasonably accurate estimate of data transfer time to improve resource allocation & scheduling for workflows and enable end-to-end data transfer optimization, we apply machine learning methods to develop predictive models for data transfer times over a variety of wide area networks. To build and evaluate these models, we use 201,388 transfers, involving 759 million files totaling 9 PB transferred, over 115 heavily used source-destination pairs ("edges") between 135 unique endpoints. We evaluate the models for different retraining frequencies and different window size of history data. In the best case, the resulting models have a median prediction error of $\leq$21% for 50% of the edges, and $\leq$32% for 75% of the edges. We present a detailed analysis of these results that provides insights into the cause of some of the high errors. We envision that the performance predictor will be informative for scheduling geo-distributed workflows. The insights also suggest obvious directions for both further analysis and transfer service optimization.

1 Introduction

With increasing data generation rates, wide-area data transfer is becoming inevitable for many science communities [1–3]. Wide-area data transfer is an important aspect of distributed science. An accurate estimation of data transfer time will significantly help both resource selection and task scheduling of geographically distributed workflows (e.g., one the cloud) [4–7]. Typically, wide-area transfers involve a number of shared resources including the storage systems and data transfer nodes (DTNs) [8] at both the source and destination, and the multi-domain network in between. Even for transfers between a user's machine and an experimental, analysis, or archival facility, the resources at the facility and the network connection are shared. Because of the fluctuation in the load on various shared resources, the data transfer rates can fluctuate significantly [9]. A number of attempts have been made to model and predict wide-area data transfer performance [10–15]. Many of these studies use a data-driven approach; some have used analytical models of individual components to predict transfer performance in dedicated environments [16]. Building an end-to-end model using the analytical models of individual components in a shared environments is challenging because the behavior of individual components can vary in an unpredictable fashion, due to competing load.

While a variety of tools are used for high-performance wide-area data transfers, such as GridFTP [17], bbcp [18], FDT [19], and XDD [20], the features that impact transfer performance are similar in most cases (e.g., number of TCP connections, number of network and disk I/O threads or processes, data transfer size, number of files). A data-driven model built using the data obtained from one high-speed data transfer tool should be generally applicable for all high-performance wide-area data transfers.

In this work, we apply a number of machine learning algorithms to develop models that can be used to predict wide-area data transfer times. We use the logs from the Globus transfer service [21] for our study. While this service orchestrates GridFTP or HTTP transfers between storage systems, we focus here on GridFTP transfers. GridFTP extends FTP to provide features such as high-speed, fault tolerance, and strong security. A number of features that are known to impact the transfer times, such as number of TCP connections, number of network and disk I/O threads, transfer size, and number of files are used to build machine-learning models.

The rest of this paper is organized as follows. First, in §2, we present background and motivation for building the predictor as well as the data we used in this paper. Next, in §3 we introduce the features we used to build the machine learning based predictor. A brief introduction of the machine learning algorithms as well as their the hyperparameters tuning, and model selection are given in §4. Then, we describe experiments, analyze results, and study feature importance in §5. Further insights into prediction errors are given in §6. Finally, we review related work in §7, and summarize our conclusions and briefly discuss future work in §8.

2 Background, motivation, and data

2.1 Background

To understand the characteristics of wide-area data transfer throughput, we analyzed the Globus transfer logs of the 1,000 source-destination pairs ("edges") that transferred the most bytes. Figure 1 shows the relative standard deviation, also known as coefficient of variation—a measure of variability relative to the mean—of the throughput for these edges. We observe a large variation in throughput for many of these edges, which reiterates the fact that predicting wide-area transfer performance is a challenging task.

In previous work [16], we characterized the features that impact wide-area data transfers and used linear and nonlinear models to validate the explainability of the features. Our goal in that work was to *explain* the performance of wide-area transfers through interpretable models. Here, we focus on applying machine learning algorithms to *predict* transfer performance.

2.2 Motivation

The primary motivation of this work is to provide a reasonably accurate estimate of data transfer time so that workflow management systems (e.g., pegasus [22]) can allocate appropriate resources for distributed science workflows and schedule the tasks more efficiently. In general, an accurate data transfer performance prediction model can be useful for many purposes, including the following:

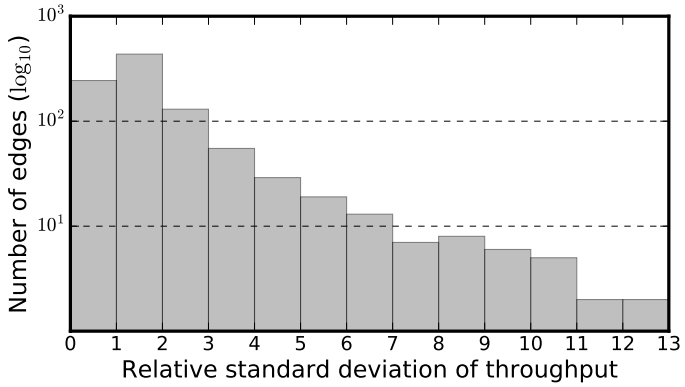


Fig. 1: Relative standard deviation (standard deviation divided by the mean) for the top 1,000 heavily used edges in Globus

1. It can be used for resource selection (when computing resource are available on multiple HPC centers or regions of cloud), and scheduling (e.g., reserve computing resource beforehand) of distributed workflows [4, 5].
2. Choosing appropriate tunable parameters of data transfer tools plays a crucial role in data transfer time [3, 23, 24]. A model that captures the influence of tunable parameters under different circumstances can be used to identify optimal tunable parameters of the transfer tools under a given condition (i.e., dataset characteristics and external load), e.g., by allowing for exhaustive search over parameters without adding real load to the infrastructure [24–27].
3. They can be used to improve user satisfaction, by setting expectations at the start of the transfer. Users can also plan their job accordingly based on the prediction.

2.3 Data

For this study, we use Globus transfer logs from 2016/01/01 to 12/31/2017, which contains 2.7 million records. We discard transfers with elapsed time shorter than 10 seconds, because we do not see it as useful to predict these small transfers in practice; transfers that were canceled by users; and transfers from Globus tutorial endpoints, leaving 0.9 million transfers, which involve 24,951 unique source-destination pairs. The top 10% of the 24,951 edges contribute 83.6% of the transfers. Only 607 edges have more than 100 transfers. Figure 2 shows the distribution of the number of transfers over edges. We consider edges with more than 150 transfers as *heavily used*.

To evaluate the models for online use, we set aside transfers from May 1, 2017 onwards for testing, and use only the transfers before this date for tuning the hyperparameters of machine learning algorithms, model selection, and training selected machine learning models: 201,388 transfers over 115 heavily used source destination pairs (involving 135 unique endpoints), totaling 9 PB in 759 million files. In terms of round trip time (RTT, a key difference among edges) between source and destination endpoints, the 50th and 75th percentile of these 115 edges are 2.46ms and 18.63ms respectively, compared with 2.37ms and 19.40ms for all edges, respectively. We believe that these 115 edges are sufficiently representative to capture the potential behavior of our models on other edges that are heavily used but lack sufficient transfers before and after May 1, 2017 to be considered in this paper.

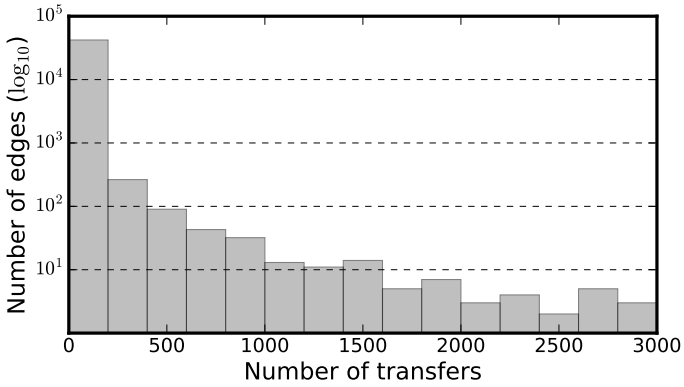


Fig. 2: Distribution of the number of transfer over edges.

3 Transfer features

Feature engineering is a key data preparation and processing step in machine learning. Often it is a manual process of transforming raw features into derived features that better represent the problem structure to the predictive models, resulting in improved model accuracy on testing data. This process involves constructing new features from raw features by combining or splitting existing features into new ones, often by the application of domain knowledge.

In our previous work [16], we employed 16 features for explaining wide-area data transfer performance, in three groups: tunable parameters specified by users, transfer characteristics that describe file properties, and competing load. We extracted the features in the first two groups directly from Globus transfer logs, but constructed those in the third group to capture resource load conditions on endpoints. Specifically, we created profiles for endpoint CPU load and network interface card load via feature engineering and showed that these profiles can be used to explain a large fraction of transfer performance in many cases. Besides the number of transfer faults, which is known to us only after the transfer, the other 15 features can be used for predicting transfer performance.

Here we expand this set to include five more features: four representing the wide-area network outbound and inbound bandwidth contention for a transfer at the transfer's source and destination institutions, and one representing pipeline depth, a tunable parameter that Globus uses to improve the performance of transfers consisting of many small files. Table 1 lists all the features (lower 20 terms) and some of the notation (the first seven terms) used in this article.

In previous work [16], we took into consideration competing transfers at a transfer's source and destination endpoints. But overlapping transfers involving other endpoints at the same source/destination institution may also compete with a given transfer for resources in the last mile. In an attempt to capture this contention, we used the MaxMind IP geolocation service [28] to obtain the approximate location of endpoints and used that information to estimate competing load based on proximity. We considered the throughput of simultaneous transfers within eight kilometers as competing transfers to the transfer of interest.

We introduce four features to quantify the wide-area network contention from the same institution. Assume that there is a transfer k from institution I_{src} to institution

Table 1: Notation used in this article. The lower 20 terms are used as features in our machine learning algorithms, of which the first 15 are from Liu et al. [16] and the remaining five are developed in this paper.

src_k	Source endpoint of transfer k.
dst_k	Destination endpoint of transfer k.
I_k^{src}	Institution of the source endpoint of transfer k.
I_k^{dst}	Institution of the destination endpoint of transfer k.
Ts_k	Start time of transfer k.
Te_k	End time of transfer k.
R_k	Average transfer rate of transfer k.
K^{sin}	Contending incoming transfer rate on src_k.
K^{sout}	Contending outgoing transfer rate on src_k.
K^{din}	Contending incoming transfer rate on dst_k.
K^{dout}	Contending outgoing transfer rate on dst_k.
C	Concurrency: Number of GridFTP processes.
P	Parallelism: Number of TCP channels per process.
S^{sin}	Number of incoming TCP streams on src_k.
S^{sout}	Number of outgoing TCP streams on src_k.
S^{din}	Number of incoming TCP streams on dst_k.
S^{dout}	Number of outgoing TCP streams on dst_k.
G^{src}	GridFTP instance count on src_k.
G^{dst}	GridFTP instance count on dst_k.
Nf	Number of files transferred.
Nd	Number of directories transferred.
Nb	Total number of bytes transferred.
Q^{sin}	Contending incoming transfer rate on I_k^{src}.
Q^{sout}	Contending outgoing transfer rate on I_k^{src}.
Q^{din}	Contending incoming transfer rate on I_k^{dst}.
Q^{dout}	Contending outgoing transfer rate on I_k^{dst}.
D	Pipeline depth.

I_{dst}. The Globus **contending transfer rate** for a transfer k at that transfer's source institution (I_{src}) and destination institution (I_{dst}) endpoints is as follows:

$$Q^{x\in\{sout,sin,dout,din\}}(k) = \sum_{i\in A_x} \frac{\mathcal{O}(i,k)}{Te_k - Ts_k} R_i, \tag{1}$$

where *sout* and *sin* denote outgoing and incoming at institution I_{src}, respectively, and *dout* and *din* represent outgoing and incoming at institution I_{dst}, respectively; A_x is the set of transfers (excluding k) with I_{src} as source, when $x = sout$; I_{src} as destination, when $x = sin$; I_{dst} as source, when $x = dout$; and I_{dst} as destination when $x = din$; and R_i is the throughput of transfer i, and $\mathcal{O}(i,k)$ is the overlap time for the two transfers:

$$\mathcal{O}(i,k) = \max\left(0, \quad \min(Te_i, Te_k) - \max(Ts_i, Ts_k)\right).$$

Pipelining, D, speeds transfers involving many small files by dispatching up to D FTP commands over the same control channel, back to back, without waiting for the first command's response [3]. This method reduces latency and keeps the GridFTP server constantly busy since it is never idle waiting for the next command.

Although we performed this work using Globus data, we believe that our methods and conclusions are applicable to all wide-area data transfers. The reason is that the features we used (number of TCP connections, number of network and disk I/O threads / processes, data transfer size, number of files, competing load) are generic features that impact the performance of any wide-area data transfer, irrespective of the tool employed. The pipeline feature is applicable to all tools that involve a command response protocol. The data used in this paper are publicly available at `https://github.com/ramsesproject/wide-area-data-transfer-logs`.

4 Machine-learning-based modeling

We use supervised learning, a class of machine learning (ML) approach, to model throughput as a function of input features. The success of the supervised-learning approach depends on several factors such as identifying appropriate data-preprocessing techniques, selecting the best supervised-learning algorithm that performs well for the given training data, and tuning the algorithm's hyperparameters. Figure 3 illustrates the process we use to obtain the best model for each edge. We detail these steps next.

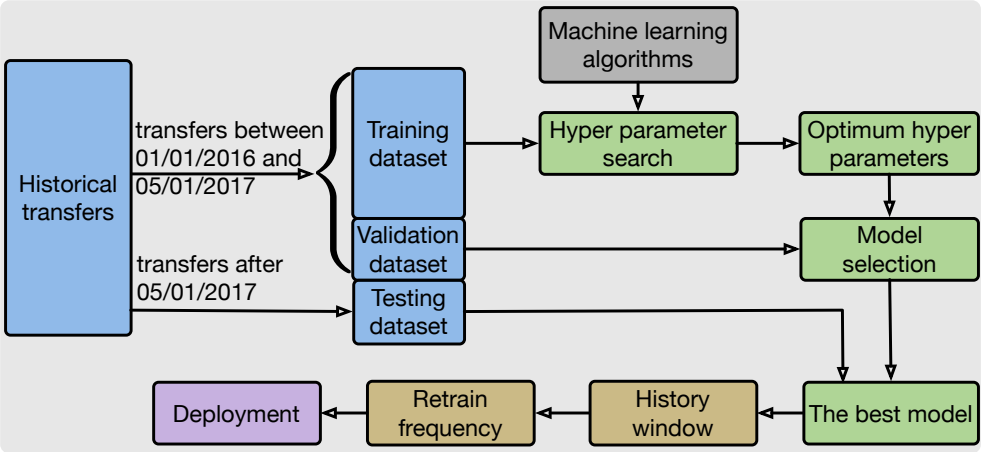


Fig. 3: Model selection process

4.1 Preprocessing

Many ML algorithms are more effective if the features have the same scale and the training data set is standardized. We apply scale transformation, which involves computing the standard deviation for each feature and dividing each value in that feature by that standard deviation. We also observed that the variance of the measured throughput is large for several endpoints. Since several ML algorithms minimize the mean squared error on the training set, if we use raw throughput, then the model fitting is biased towards large throughputs values but not the small ones. To avoid this, we applied logarithm transformation for the throughput.

4.2 Supervised learning algorithms

Many supervised learning algorithms exist in the ML literature. They can be categorized as regularization, instance-based, kernel-based, bagging, and boosting algorithms. Typically, the best algorithmic choice depends on the type of relationship between the input features and the throughput. Instead of focusing on a single ML algorithm for our study, we selected a set of ML algorithms of increasing complexity. We start from linear regression, the most basic ML algorithm. We adopt ridge regression that addresses overfitting problem of linear regression using regularization. Ensemble methods build a number of simple models and combine them to obtain better predictive performance and they are shown to obtain state-of-the-art results in a wide range of predictive modeling problems [29]. Two families of ensemble methods are (1) *bagging* that build several simple models independently and then average their predictions—Bagging, ExtraTrees, and Random Forest algorithms are selected from this class; (2) *boosting* that build simple models incrementally to decrease the training error in a sequential way—GradientBoosting, AdaBoost, and eXtreme Gradient Boosting algorithms are selected from this class. We also evaluated support vector machines as a candidate for a kernel-based regression method, but found that it did not perform well and so we omitted it from the study. In the rest of the section, we provide a high-level overview of the algorithms considered in our study.

Multivariate linear regression tries to find $h(x)$ by fitting a linear equation $h(x) = c + \sum_{i=1}^{M} \alpha^i \times x^i$, where c is a constant and α^i is the coefficient of the input x^i. Appropriate values of (c, α) are obtained by the method of least-squares, which minimizes the sum of the squared deviations between the y_i and $h(x_i)$ in $\mathcal{D}$.

Ridge regression (RR) [30] is a regularization algorithm that addresses the overfitting problem faced by linear regression. Overfitting occurs when the model becomes sensitive to even small variations in the training set output and lose prediction accuracy on the testing and validation set. To avoid this problem, RR, in addition to minimizing the error between predicted and actual observations, penalizes the objective by a factor α of the input coefficients. Here, $\alpha \geq 0$ is a user-defined parameter that controls the tradeoff between minimizing error and minimizing sum of square of coefficients.

Bagging trees (BR) [31] builds `nt` decision trees. Each decision tree is obtained on a random subsample of the original dataset by recursively splitting the multi-dimensional input space into a number of hyper-rectangles such that inputs with similar outputs fall within the same rectangle. Given a new point x^* to predict, each tree follows if-else rule and returns the constant value at the leaf as the predicted value. The mean of all the predicted values will be returned as the final prediction value.

Random forest (RFR) [32,33] is also a bagging approach that builds `nt` decision trees similar to BR but at each split, RFR considers only a random subset of inputs and selects the best one for split based on mean squared error.

Extremely randomized trees (ETR) [34] differs from RFR in the way in which the splits are computed. As in RFR, a random subset of inputs is considered for the split, but instead of looking for the best mean squared error, the values are drawn at random for input and the best of these randomly-generated values is picked as the splitting rule.

Adaptive boost regressor (ABR) [35] is a tree-based boosting approach that differs from tree-based bagging in the way it builds the trees and aggregates the

results: `nt` trees are built sequentially such that the m^{th} tree is adjusted to correct the prediction error of the $(m-1)^{th}$ tree. Each sample point in the training set is assigned a weight proportional to the current error on that point so that subsequent tree focuses on points with relatively higher training errors. Given a new test point x^*, each tree calculates a predicted value. These predicted values are weighted by each tree's weight, and their sum gives the prediction for the ensemble model.

Gradient boosting trees (GBR) [36] are similar to ABR, in which `nt` trees are built sequentially. The key difference is that GBR tries to minimize squared error loss function by using the residuals of the $(m-1)^{th}$ model as the negative gradient. It generates a new model at the m^{th} iteration by adding the m^{th} tree that fits the negative gradient to the $(m-1)^{th}$ model.

eXtreme Gradient Boosting (XGB) [37] is a high-performing gradient boosting optimized software framework and implementation, which is used widely by data scientists to achieve state-of-the-art results on many ML challenges such as Kaggle and KDDCup. XGB is similar to GBR but the former adopts a more regularized model formalization than the latter which gives it better performance. Moreover, XGB uses several novel algorithmic optimizations—a novel approximate tree learning algorithm, an efficient procedure to handle training point weights in approximate tree learning, and system level optimization—out-of-core computation to process data that is too large to fit in main memory and cache-aware learning to improve speed and scalability.

4.3 Hyperparameter tuning

Many ML algorithms require user-defined values for hyperparameters that strongly influence the performance of the algorithm. These parameters include not only categorical but also numerical parameters such as α, number of trees, and maximum tree depth. Choosing appropriate parameter settings can have a significant impact on the performance. Unfortunately, finding performance-optimizing parameter settings is data dependent and difficult task.

This task has traditionally been tackled by hand, using a trial-and-error process and/or grid search. In the past few years, new algorithmic methods have begun to emerge to address these serious issues. In particular, a random search algorithm has shown to be more effective than grid search for tuning the hyperparameters of ML algorithms [38]. We consider a random search over the feasible domain D without replacement. This consists in sampling n_s random parameter configurations for a given supervised learning algorithm, evaluate each of them on the training set by cross validation, and selecting the best according to a user-defined metric (e.g., the median absolute percentage error).

We tuned the hyperparameters for each learning algorithm by using a random search that samples $n_s = 200$ configurations and selected the best based on cross validation on the training data. The parameters of each algorithm considered for tuning are as follows. **XGB**: The number of boosted trees to fit, the maximum tree depth for base learners, the minimum loss reduction required to make a further partition on a leaf node of the tree, the subsample ratio of the training instance, and the subsample ratio of columns when constructing each tree. **BR**: The number of base estimators in the ensemble, the number of samples to train each base estimator, and whether samples are drawn with replacement. **GBR**: The number of boosting stages to perform, the maximum depth of the individual regression estimators, and the learning rate. **RFR** and **ETR**: The number of trees in the forest, the maximum depth of the

tree, and whether bootstrap samples are used when building trees. **ABR**: The maximum number of estimators at which boosting is terminated, and the learning rate. **RR**: The regularization strength. Note that linear regression does not have tunable parameters.

4.4 Model selection

For each edge, we select the best model as follows. First, as shown in Figure 3, we split transfers before 05/01/2017 as two sets: the first 70% of the transfers as training set and the remaining 30% as validation set. For each supervised learning algorithm, we run hyperparameter optimization with $n_s = 200$, 10-fold cross validation on the training set, and root mean squared error as the metric that needs to be minimized. This is repeated for each algorithm. Once we obtain the best hyperparameters from each of the seven ML algorithms, we predict the throughput on the validation set. The best model for the edge is the one that has the minimal prediction error on the validation set.

Figure 4 shows the overall accuracy of each algorithm on the validation set, in which the median absolute percentage error (MdAPE) is used to represent model accuracy. The **Best** in the plot is the accuracy when each edge uses the best algorithm. From the results, we can observe that **XGB** works the best for most number

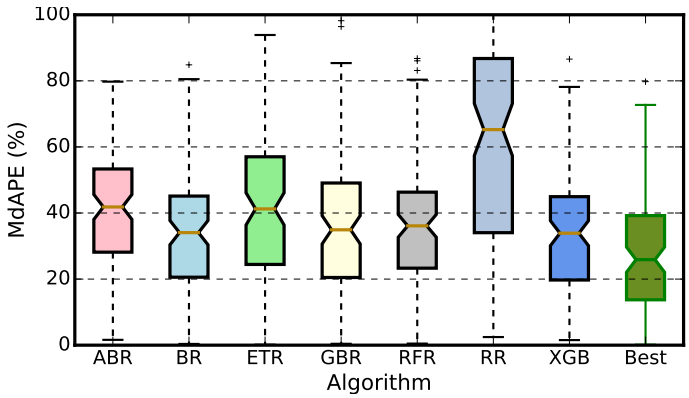


Fig. 4: Validation results. *Best* represent validation errors when the best algorithm is used for each edges.

of edges; **GBR** and **BR** seem to have similar accuracy. However, the best case obviously outperforms any single algorithm. Table 2 presents the statistics on the winning algorithms.

5 Experimental results

After identifying the best model for each edge, we search hyperparameters again and retrain the model with the combined training and validation data sets. Then, we evaluate model accuracy on the test data set (i.e., transfers between 05/01/2017 and 12//31/2017). From a practical deployment perspective, we focus on the following three issues:

Table 2: Statistics of the best model selection

Algorithm	**Pairs**
GradientBoostingRegressor	23
Ridge	6
XGBRegressor	31
BaggingRegressor	18
AdaBoostRegressor	9
RandomForestRegressor	14
ExtraTreesRegressor	14

1. How many historical instances are needed to train the model? That is, how many old instances can be abandoned?
2. What is the appropriate retraining frequency to deal with changes because of software or hardware upgrades?
3. Do we have to take into account the temporal aspect of the data? That is, can we randomly choose K transfers or should we use the most recent K transfers for model training.

5.1 Size of the training data

The time required to retrain the model scales with the number of data points considered. Furthermore, since endpoint performance can be expected to change over time, for example because of software or hardware upgrades, it is necessary to remove the data points before such changes. But identifying the point at which an upgrade or other change occurs may not always be straightforward in the online setting. Therefore, we empirically evaluate the volume of historic data required to make a reasonable prediction. Here, we use the most recent w ($w \in \{25, 50, 100, 500, 1000, 1500, 2000\}$) transfers before May 1, 2017, to train the model, and we test each trained model on all transfers from May 1, 2017 to December 31, 2017. Figure 5 compares quantile distribution of MdAPE under different window sizes for training and the corresponding model training time. In the plot, *Q25*, *Q50* and *Q75* represent 25th, 50th and 75th quantile of MdAPE separately. *RCT* and *RND* denotes most recent N_{train} transfers and randomly chosen N_{train} transfers, respectively. *All* means that we used all transfers before May 1, 2017, to train the model.

Models trained with random data points are slightly better than models trained with the most recent data points; in most cases this performance improvement is only a little (except the $w = 25$ case). The results show that increasing the values of w decreases the error but only until $w = 1000$. The observed improvements become insignificant for $w > 1000$. However, the model training time increases significantly when the window size is increased (this is true for all cases except when w is increased from 25 to 50 and from 1000 to 1500). These results indicate that most recent 500 to 1,000 data points is sufficient for training the model.

We use the methods that users would use in practice, specifically, *average*, *median* and the performance of the *previous* transfer, as the baselines. Specifically, for a given new transfer, there are w historical transfers over the same edge, we consider the median performance, average performance and the performance of the most recent one transfer as baseline predictions. Figure 6 shows the 50th quantile of MdAPE for these three baseline predictors.

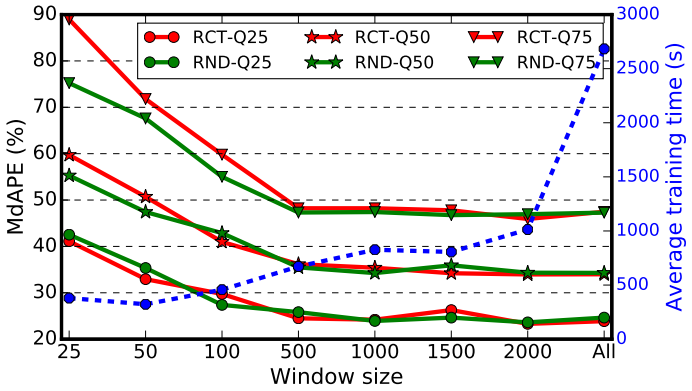


Fig. 5: Prediction errors (solid lines) and model training time (dotted blue line) as a function of the number of transfers (N_{train}) used to train the model. **Q25**, **Q50** and **Q75** represent 25th, 50th and 75th quantile of MdAPE separately. **RCT** and **RND** denotes most recent N_{train} transfers and randomly chosen N_{train} transfers individually. **All** means that we used all transfers before May 1, 2017, to train the model.

In comparison with Figure 5 (*RCT-Q50* and *RND-Q50*), these three baseline predictors have significantly worse prediction errors than our machine learning based predictor.

5.2 Retraining frequency

We evaluated and compared the results obtained when retraining models every 3 months, 1 month, 2 weeks, 1 week, 2 days, and online. To this end, we split the test dataset into multiple datasets based on the frequency. For the 3 month frequency, we train the model using all transfers before May 1, 2017, and evaluate it with transfers from May 1, 2017, to July 31, 2017. Then, we retrain model using data before August 1, 2017 and evaluate it with the transfer from August 1, 2017 to October 31, 2017, and so on. Similarly, for the monthly frequency, we evaluate the model trained using all transfers before May 1, 2017 with the transfers in May, 2017. Then we retrain the model using all the data before June 1, 2017, and evaluate it with transfers in June, 2017, and so on. We follow a similar procedure for other retraining frequencies. For online retraining, we monitor the prediction error of new transfers and retrain the model with the most recent w transfers if the MdAPE of the recent k transfers is larger than a certain threshold. In this study, we set w, k, and threshold to 100, 4, and 15%, respectively. We also experimented with w, k, and threshold of 500, 10, and 15%, respectively, but those values did not result in a significant difference.

Figure 7 compares the MdAPE distribution on all edges for different retraining frequencies. We observe that more frequent retraining results in lower errors and that online retraining achieves the lowest error.

5.3 Feature importance

To gain insights into the impact of the features listed in Table 1 on the throughput, we analyze their importance using the relative feature importance capability of XGB [37].

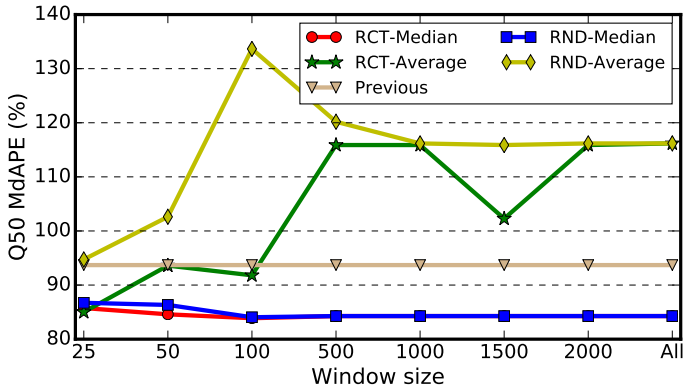


Fig. 6: The prediction error (50th quantile MdAPE) when use median, average and previous transfer as performance predictor.

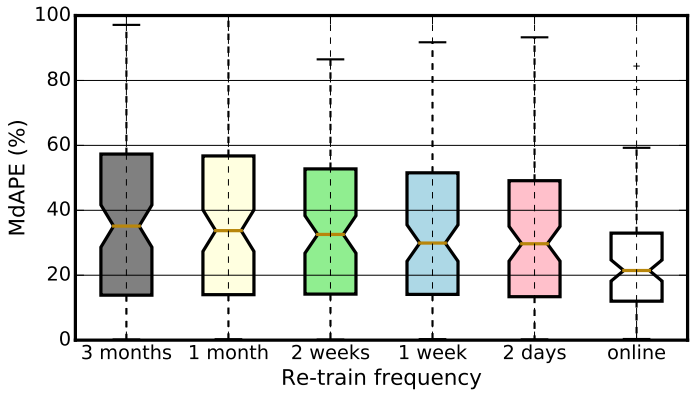


Fig. 7: Retrain frequency versus prediction error on testing dataset.

First, we compute the relative importance of the features on each edge and select the five most important ones. Then we tabulate the frequency of the top five in all the edges in Figure 8.

As one can see from Figure 8, transfer size Nb and K^{sout}, K^{din} and S^{sout} which have contention in the same direction with the transfer of interest, are important for most of the endpoint pairs. The four new introduced features in this paper (Q^{sout}, Q^{din}, Q^{sin} and Q^{dout}), which quantify contention from simultaneous transfers from the same institution (within eight kilometers), are also important.

6 Further insights

We hypothesize that unknown load is the cause of the high prediction error for some of the endpoint pairs. Here we examine the effect of unknown load and study transfers with less unknown load.

6.1 Impact of unknown load

Three factors may affect the data transfer rate: user-specified tunable parameters, transfer characteristics, and competing load [16, 24]. To validate our hypothesis con-

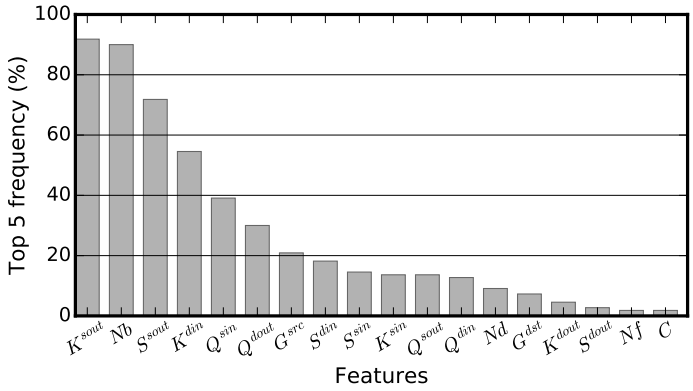


Fig. 8: Frequency of features that appear in the top 5 most important features of each edge model.

cerning unknown external load, we consider transfers from Argonne to Fermilab. The reasons for selecting this endpoint pair are twofold: (1) it has a sufficient number of transfers with similar file characteristics (number of files, directories, and average file size) and the same tunable Globus parameters; and (2) we observe quite different rates for these transfers. Since transfer parameters and characteristics are identical, the transfer rate should be affected only by competing load.

We split the transfers over this edge into three groups:

1. Transfers with rate greater than 50% of the maximum rate observed over this endpoint pair. These transfers are likely to have less contending load.
2. After filtering out the transfers in group (1), for each remaining transfer, we compute the aggregate outgoing rate on source endpoint during the transfer by summing the rate of the transfer under consideration and the outgoing rates for other overlapping transfers at the source endpoint ($R + K^{sout}$). Similarly, we compute the aggregate incoming rate on destination endpoint during each transfer by summing the rate of the transfer under consideration and the incoming rates of other overlapping transfers at the destination endpoint ($R + K^{din}$). We sort the transfers in descending order by their source endpoint's aggregate outgoing rate and take the top 5%. Then, we sort the remaining transfers in descending order of their destination endpoint's aggregate incoming rate and take the top 5%. These transfers that form group (2) have significant known contending load but less unknown contending load among the transfers compared with group (1).
3. We apply the same procedure that created group (2), but extract bottom 5% on source and destination endpoints. These transfers are likely to have high unknown contending load because their known load is less and throughput is low as well.

As a result, we get 161 transfers in group (1), 175 in group (2), and 175 in group (3) (shown in Figure 9). We split each group by 70% and 30% for training and testing respectively. Table 3 shows the prediction errors with different algorithms.

Group 1 and 2 have low prediction error with nonlinear models (the second and third model in Table 3) and group 3 always have higher error than the first two groups. The linear model works well for group 1 but not for group 2 or 3. The poor prediction accuracy of linear model and the high prediction accuracy of machine learning models in group 2 indicate nonlinear relationship between 'known load' and 'transfer rate', which is consistent with findings in [16].

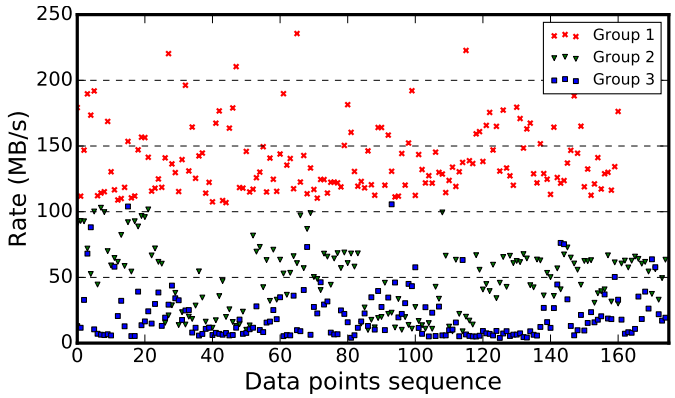


Fig. 9: Group transfers by their rate and known load.

Table 3: Prediction error with different machine learning algorithm on the three groups.

Algorithm	**Group**	**Q50**(%)	**Q75**(%)	**Q90**(%)
Ridge Regression	1	11.24	18.19	22.63
	2	20.04	33.08	64.33
	3	35.37	126.54	223.29
XGBRegressor	1	11.85	22.91	25.20
	2	8.20	18.06	29.36
	3	27.16	51.02	72.49
BaggingRegressor	1	9.54	18.83	25.02
	2	9.46	14.81	32.64
	3	29.85	51.27	133.48

6.2 Transfers with fewer unknowns

Since transfers with high unknown load is the source of noise when we use them to train the machine learning model, here we trained the models using transfers with less unknown load and study how they perform. We compute the relative 'known load (KL)' for a transfer as the ratio of aggregate transfer rate to maximum observed aggregated throughput. For the source endpoint of the transfer of interest, this is given by:

$$KL_k^{src} = \frac{K^{sout} + R_k}{DR^{max}}, \tag{2}$$

where DR^{max} is the maximum aggregated outgoing throughput observed from the source endpoint. Similarly, for the destination endpoint of the transfer of interest, we have:

$$KL_k^{dst} = \frac{K^{din} + R_k}{DW^{max}}. \tag{3}$$

where DW^{max} is the maximum aggregated incoming throughput observed from the destination endpoint. We then define the relative load of a transfer k as:

$$KL_k = \max\left(RL_k^{src}, RL_k^{dst}\right). \tag{4}$$

Intuitively, KL_k measures the fraction of bandwidth usage that has been observed from Globus transfers. A higher value means less unknown (non-Globus) contending

load, as the total bandwidth available at the source and destination is fixed and the majority of the capacity usage is known. Thus in this section, we use

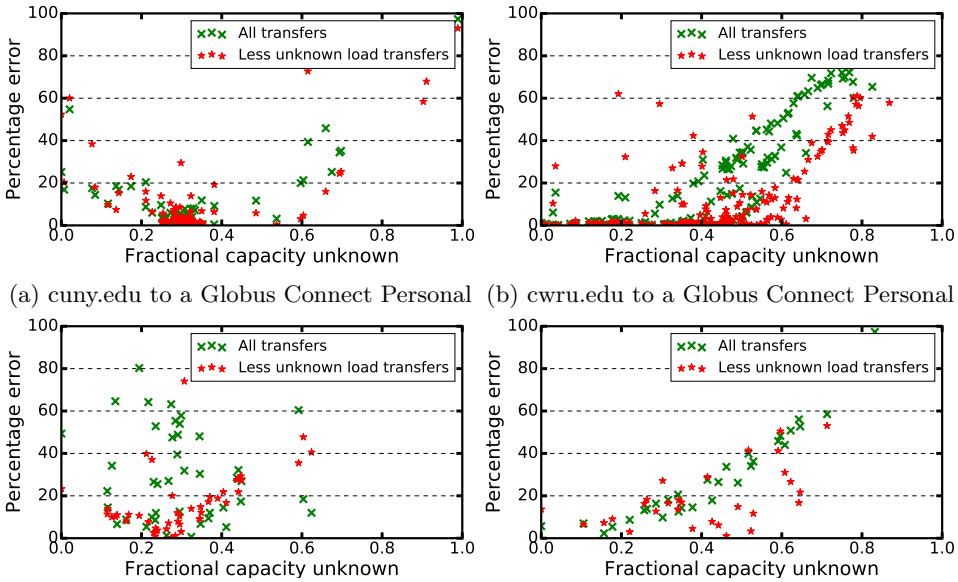
$$UC_k = 1 - KL_k \tag{5}$$

to represent the 'fractional capacity unknown' to the transfer. A larger value means that the transfer is more likely to have experienced unknown load.

In order to verify the influence of unknown load on the model being trained, we use only transfers with $UC_k \leq 0.5$ to train the model. Specifically, we ignore transfers with $UC_k > 0.5$ in the training data set as these transfers are more likely to have high unknown contending load.

Then, we use transfers with $UC_k \leq 0.5$ to train the model and test the trained model on the *whole* testing set (i.e., transfers between 05/01/2017 and 12/31/2017. We infer that the model trained with these clean data is more representative of transfers with less unknown load because there is less noise in the training set. Thus, we expect this model to make better predictions for transfers with less unknown load.

Figure 10 compares the relationship between 'fractional capacity unknown' and the absolute percentage error. The green cross points are prediction errors for the model that was trained with all transfers before 05/01/2017 (irrespective of the 'fractional capacity unknown') and red star points represent prediction errors for the model that was trained by using transfers before 05/01/2017 and with $UC_k \leq 0.5$.



(a) cuny.edu to a Globus Connect Personal

(b) cwru.edu to a Globus Connect Personal

(c) harvard.edu to a Globus Connect Personal

(d) westgrid.ca to a Globus Connect Personal

Fig. 10: Comparison of the 'fractional capacity unknown' of a transfer versus absolute performance prediction error (%), for two models: one trained only with less unknown load transfers and the other trained with all transfers. We show results for four different source endpoints. We note that the destination endpoints in the figures are also different from each other.

As shown in Figure 10, when we look at the prediction errors for either model i.e., the model trained with all transfers (green cross points) or the model trained only by using transfers with $UC_k \leq 0.5$ (red star points), we observe that the transfers with lower 'fractional capacity unknown' get more accurate prediction. The reason is that these transfers are less likely to have high unknown load. When we compare the prediction error of the same transfer, i.e. a green point and a red point with the same UC_k, model that was trained with cleaner data (marked by red star points) make more accurate predictions than the model that was trained with all transfers (marked by green cross points), when UC_k is less than 0.5. However, we cannot use this method to improve the model accuracy because the relative known load is not known to us before the transfers are done. We use this analysis to show the influence of external (unknown) load in training as well as in prediction. To improve the prediction, we need to develop methods that can measure the external load online.

7 Related work

Several researchers have developed regression-based throughput prediction models using historical data. Vazhkudai et al. [39] developed mean, median, autoregressive, sliding window, and rule-based approaches to predict the performance of wide area data transfers. Swany et al. [40] developed a multivariate time-series forecasting technique to enable Network Weather Service, a TCP throughput prediction framework, to automatically correlate monitoring data from different sources, and to exploit that observed correlation to make better, longer-ranged forecasts. Lu et al. [41] developed TCP throughput predictions based on an exponentially weighted moving average of bandwidth probes. He et al. [42] studied analytical and empirical models for predicting the throughput of large TCP transfers. Their empirical approach used standard time series forecasting techniques to predict throughput based on a history of throughput measurements from previous TCP transfers on the same path. Huang et al. [43] developed time series of windows of segments arriving at the receiver, and predicted future throughput by exploiting knowledge of how TCP manages transfer window size. When the file transfer time series resembles a known TCP pattern, this information is used for prediction, otherwise simple heuristics are used.

A central theme in all of the aforementioned works is that the problem is tackled from a time-series perspective. There is always a temporal aspect to the prediction: in order to predict the future throughputs accurately, the approach requires immediate past throughputs. Our approach, in contrast, treats the throughput prediction from a static modeling perspective. Consequently, we can use historical data to predict throughput as long as the relationship between inputs and outputs does not change.

A number of previous works have treated throughput prediction without temporal dependency. Shah et al. [44] developed an artificial neural network model for TCP throughput estimation based on loss event rate, round trip time, retransmission time out, and number of packets acknowledged. Our work deals with application-level end-to-end WAN transfer and considers features at the endpoint level. Mirza et al. [45] developed a support vector regression approach to predict throughput of TCP transfers in a controlled laboratory setting. Their approach uses prior file transfer history and measurements of endpoint pair network path properties as a training data for throughput prediction. Kettimuthu et al. [46] developed linear models that can control bandwidth allocation. They showed that log models that combine total source

concurrency, destination concurrency, and a measure of external load are effective. Nine et al. [47] analyzed historical data consisting of 70 million file transfers and applied data mining techniques to extract the hidden relations among the parameters and the optimal throughput. They used neural networks and support vector machines to develop predictive models and to find optimal parameter settings. Our approach considers a broad range of features, several high performing supervised learning algorithms, and adopts a rigorous model selection methodology in which the best is automatically selected based on the training data.

Arslan et al. [23] developed the HARP auto-tuning system, which uses historical data to derive endpoint-pair-specific models based on protocol parameters. By running probing transfers, HARP captures the current load on the network, which is then augmented with a polynomial regression model to increase the predictive accuracy. Moreover, we consider several models and choose the best automatically to hedge against the poor performance of any single method. Hours et al. [48] introduced the use of copula, a statistical approach to model the dependence between random variables, to model the relationship between network parameters and throughput as multidimensional conditional densities. While this method is orthogonal to the approach we used, it is limited by the data assumptions and presence of outliers.

Liu et al. [49] conducted a systematic examination of a large set of data transfer logs to characterize transfer characteristics, including the nature of the datasets transferred, achieved throughput, user behavior, and resource usage. Their analysis yielded new insights that can help design better data transfer tools, optimize networking and edge resources used for transfers, and improve the performance and experience for end users. Liu et al. [50] analyze various logs pertaining to wide area data transfers in and out of a large scientific facility. That comprehensive analysis yielded valuable insights on data transfer characteristics and behavior, and revealed inefficiencies in a state-of-the-art data movement tool.

8 Discussion and conclusion

Wide-area data transfer is an important aspect of distributed science. Faster delivery of data is important but predictable transfers are equally (or even more) important. We developed machine learning models to predict the performance of wide area data transfer. We also evaluated them for practical deployment. We believe that our evaluation strategy will serve as a reference to evaluate machine learning models for practical deployments not just for predicting the performance of wide area data transfers but also for other prediction use cases. We showed that our models perform well for many transfers, with a median prediction error of $\leq$21% for 50% of source-destination pairs ("edges"), and $\leq$32% for 75% of the edges. For other edges, errors are high, an observation that we attribute to unknown load on the network or on endpoint storage systems. We also showed that unknown load can interfere with model training and that eliminating transfers with high unknown load from training data can improve prediction accuracy for transfers with less unknown load. Thus, collecting more information about endpoint load can further improve the prediction accuracy.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. We gratefully acknowledge

the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

References

1. R. Kettimuthu, G. Agrawal, P. Sadayappan, and I. Foster, "Differentiated scheduling of response-critical and best-effort wide-area data transfers," in *2016 IEEE International Parallel and Distributed Processing Symposium*, May 2016, pp. 1113–1122.
2. W. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, "Data management and transfer in high-performance computational grid environments," *Parallel Comput.*, vol. 28, no. 5, pp. 749–771, May 2002. [Online]. Available: http://dx.doi.org/10.1016/S0167-8191(02)00094-7
3. R. Kettimuthu, Z. Liu, D. Wheeler, I. Foster, K. Heitmann, and F. Cappello, "Transferring a petabyte in a day," *Future Generation Computer Systems*, vol. 88, pp. 191 – 198, 2018. [Online]. Available: https://doi.org/10.1016/j.future.2018.05.051
4. G. L. Stavrinides, F. R. Duro, H. D. Karatza, J. G. Blas, and J. Carretero, "Different aspects of workflow scheduling in large-scale distributed systems," *Simulation Modelling Practice and Theory*, vol. 70, pp. 120 – 134, 2017. [Online]. Available: https://doi.org/10.1016/j.simpat.2016.10.009
5. Z. Liu, R. Kettimuthu, S. Leyffer, P. Palkar, and I. Foster, "A mathematical programming- and simulation-based framework to evaluate cyberinfrastructure design choices," in *IEEE 13th International Conference on e-Science*, Oct 2017, pp. 148–157. [Online]. Available: http://doi.org/10.1109/eScience.2017.27
6. T. Bicer, D. Gürsoy, R. Kettimuthu, F. De Carlo, and I. T. Foster, "Optimization of tomographic reconstruction workflows on geographically distributed resources," *Journal of Synchrotron Radiation*, vol. 23, no. 4, pp. 997–1005, Jul 2016.
7. R. Kettimuthu, Z. Liu, I. Foster, P. H. Beckman, A. Sim, J. Wu, W. keng Liao, Q. Kang, A. Agrawal, and A. Choudhary, "Toward autonomic science infrastructure: Architecture, limitations, and open issues," in *the 1st Autonomous Infrastructure for Science workshop*, ser. AI-science 2018. New York, NY, USA: ACM, 2018. [Online]. Available: http://doi.acm.org/10.1145/3217197.3217205
8. R. Kettimuthu, G. Vardoyan, G. Agrawal, P. Sadayappan, and I. Foster, "An elegant sufficiency: load-aware differentiated scheduling of data transfers," in *SC15: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2015, pp. 1–12.
9. S. Vazhkudai, "Enabling the co-allocation of grid data transfers," in *Proceedings. First Latin American Web Congress*, Nov 2003, pp. 44–51.
10. D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1246–1259, 2006.
11. B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, and D. Gunter, "The NetLogger methodology for high performance distributed systems performance analysis," in *7th International Symposium on High Performance Distributed Computing*. IEEE, 1998, pp. 260–267.
12. T. Kosar, G. Kola, and M. Livny, "Data pipelines: Enabling large scale multi-protocol data transfers," in *2nd Workshop on Middleware for Grid Computing*. ACM, 2004, pp. 63–68.
13. T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 2, pp. 83–91, 2003.
14. R. Wolski, "Forecasting network performance to support dynamic scheduling using the Network Weather Service," in *6th IEEE Symposium on High Performance Distributed Computing*, Portland, Oregon, 1997.

15. T. J. Hacker, B. D. Athey, and B. Noble, "The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network," in *16th International Parallel and Distributed Processing Symposium*, ser. IPDPS '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 314–. [Online]. Available: http://dl.acm.org/citation.cfm?id=645610.661894
16. Z. Liu, P. Balaprakash, R. Kettimuthu, and I. Foster, "Explaining wide area data transfer performance," in *26th ACM Symposium on High-Performance Parallel and Distributed Computing*, 2017.
17. W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The Globus striped GridFTP framework and server," in *SC*, Washington, DC, USA, 2005, pp. 54–61.
18. www.slac.stanford.edu/ abh/bbcp/, *BBCP*, 2017 (accessed January 3, 2017), http://www.slac.stanford.edu/~abh/bbcp/.
19. FDT, *FDT - Fast Data Transfer*, 2017 (accessed April 3, 2017), http://monalisa.cern.ch/FDT/.
20. B. W. Settlemyer, J. D. Dobson, S. W. Hodson, J. A. Kuehn, S. W. Poole, and T. M. Ruwart, "A technique for moving large data sets over high-performance long distance networks," in *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, May 2011, pp. 1–6.
21. K. Chard, S. Tuecke, and I. Foster, "Globus: Recent enhancements and future plans," in *XSEDE16 Conference on Diversity, Big Data, and Science at Scale*. ACM, 2016, p. 27.
22. E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus: a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.
23. E. Arslan, K. Guner, and T. Kosar, "Harp: Predictive transfer optimization based on historical analysis and real-time probing," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2016, pp. 288–299.
24. Z. Liu, R. Kettimuthu, I. Foster, and P. H. Beckman, "Towards a smart data transfer node," *Future Generation Computer Systems*, p. 10, June 2018.
25. E. Arslan, K. Guner, and T. Kosar, "HARP: predictive transfer optimization based on historical analysis and real-time probing," in *SC*, Piscataway, NJ, USA, 2016, pp. 25:1–25:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=3014904.3014938
26. E. Arslan and T. Kosar, "A heuristic approach to protocol tuning for high performance data transfers," *ArXiv e-prints*, Aug. 2017.
27. J. Kim, E. Yildirim, and T. Kosar, "A highly-accurate and low-overhead prediction model for transfer throughput optimization," *Cluster Computing*, vol. 18, no. 1, pp. 41–59, 2015.
28. www.maxmind.com, *MaxMind: IP Geolocation and Online Fraud Prevention*, 2017 (accessed April 3, 2017), https://www.maxmind.com.
29. R. Maclin and D. W. Opitz, "Popular ensemble methods: An empirical study," *CoRR*, vol. abs/1106.0257, 2011. [Online]. Available: http://arxiv.org/abs/1106.0257
30. A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.
31. L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.
32. T. K. Ho, "Random decision forests," in *3rd International Conference on Document Analysis and Recognition*, ser. ICDAR '95. IEEE, 1995, pp. 278–282. [Online]. Available: http://dl.acm.org/citation.cfm?id=844379.844681
33. L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324
34. P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.

35. Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *European conference on computational learning theory*. Springer, 1995, pp. 23–37.
36. J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.
37. T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *arXiv preprint arXiv:1603.02754*, 2016.
38. J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2188385.2188395
39. S. Vazhkudai, J. M. Schopf, and I. Foster, "Predicting the performance of wide area data transfers," in *International Parallel and Distributed Processing Symposium*. IEEE, 2001, pp. 10–pp.
40. M. Swany and R. Wolski, "Multivariate resource performance forecasting in the Network Weather Service," in *Supercomputing, ACM/IEEE 2002 Conference*. IEEE, 2002, pp. 11–11.
41. D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante, "Characterizing and predicting TCP throughput on the wide area network," in *25th IEEE International Conference on Distributed Computing Systems*. IEEE, 2005, pp. 414–424.
42. Q. He, C. Dovrolis, and M. Ammar, "On the predictability of large transfer TCP throughput," *Computer Networks*, vol. 51, no. 14, pp. 3959–3977, 2007.
43. T.-i. Huang and J. Subhlok, "Fast pattern-based throughput prediction for TCP bulk transfers," in *International Symposium on Cluster Computing and the Grid*, vol. 1. IEEE, 2005, pp. 410–417.
44. S. M. H. Shah, A. ur Rehman, A. N. Khan, and M. A. Shah, "TCP throughput estimation: A new neural networks model," in *International Conference on Emerging Technologies*. IEEE, 2007, pp. 94–98.
45. M. Mirza, J. Sommers, P. Barford, and X. Zhu, "A machine learning approach to TCP throughput prediction," *IEEE/ACM Transactions on Networking*, vol. 18, no. 4, pp. 1026–1039, 2010.
46. R. Kettimuthu, G. Vardoyan, G. Agrawal, and P. Sadayappan, "Modeling and optimizing large-scale wide-area data transfers," in *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2014, pp. 196–205.
47. M. Nine, K. Guner, and T. Kosar, "Hysteresis-based optimization of data transfer throughput," in *5th International Workshop on Network-Aware Data Management*. ACM, 2015, p. 5.
48. H. Hours, E. Biersack, and P. Loiseau, "A causal approach to the study of TCP performance," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 7, no. 2, p. 25, 2016.
49. Z. Liu, R. Kettimuthu, I. Foster, and N. S. V. Rao, "Cross-geography scientific data transferring trends and behavior," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '18. New York, NY, USA: ACM, 2018, pp. 267–278. [Online]. Available: http://doi.acm.org/10.1145/3208040.3208053
50. Z. Liu, R. Kettimuthu, I. Foster, and Y. Liu, "A comprehensive study of wide area data movement at a scientific computing facility," in *the 38th IEEE International Conference on Distributed Computing Systems*, ser. 2018 Scalable Network Traffic Analytics. IEEE, 2018.