Modeling and Optimizing Large-Scale Wide-Area Data Transfers

Rajkumar Kettimuthu*, Gayane Vardoyan*, Gagan Agrawal†, and P. Sadayappan†
*Mathematics and Computer Science Division
Argonne National Laboratory
E-mail: {kettimut, gvardoyan}@anl.gov
†Computer Science and Engineering
The Ohio State University
E-mail: {agrawal, saday}@cse.ohio-state.edu

***Abstract*—Data generated by experimental, simulation, and observational science is growing exponentially. The resulting datasets are often transported over wide-area networks for storage, analysis, or visualization. Network bandwidth, which is not increasing at the same rate as dataset sizes, is becoming a key obstacle to data-driven sciences.**

In this paper, we focus on how bandwidth allocation can be controlled at the level of a protocol such as GridFTP, in view of goals such as maintaining certain priorities or performing scheduling with specified objectives. In particular, we explore how GridFTP transfer performance can be controlled by using parallelism and concurrency. We find that concurrency turns out to be a more powerful control knob than is parallelism. For a source where most bandwidth is consumed by transfers to a small number of other destinations, we build a model for each destination's achieved throughput in terms of its concurrency and total concurrency (over GridFTP transfers) to other major destinations. We then enhance this model by including an indicator of the time-varying external load, using multiple ways to measure this external load.

We study the effectiveness of the proposed models in controlling the bandwidth allocation. After evaluating the numerous combinations of models and methods of measuring external load, we narrow in on the four best-performing ones, based on both their validation results and their applicability. After extensive testing of these four approaches, we find that they can obtain desired bandwidth allocations with a mean(median) error rate of 19.8%(13.8%), with 38% of the errors in our benchmark tests being less than 10% and 54% of them being less than 15%.

***Keywords*-wide-area data transfer; GridFTP; modeling data transfer**

I. Introduction

The amount of data generated by experimental, simulation, and observational science is growing exponentially. Climate data is projected to exceed hundreds of exabytes by 2020 [5]. Major climate sites (e.g., LLNL, ORNL) will host huge datasets, and several key sites spread across the world (UK, Germany, Australia, Japan, etc.) will distribute this data to climate researchers [5]. Cosmology projects are capturing huge amounts of observational data. For example, the Dark Energy Survey, designed to probe the origin of the accelerating universe and help uncover the nature of dark energy by measuring the 14-billion-year history of cosmic expansion with high precision, captures terabytes per night. The Square Kilometer Array, another cosmology project, will generate an exabyte every 13 days when it becomes operational. Many other fields, such as genome sequencing, geosciences, and materials science, are also generating more and more data.

While researchers may seek to analyze and store data in place, datasets must frequently be transported over wide-area networks—for example, from generation sites to remote facilities for storage, analysis, or visualization. Data movement bandwidths (including disk speeds, NIC bandwidths, and wide-area bandwidths) are not increasing at the same rate as are dataset sizes, and this situation is becoming a major constraint for data-driven sciences. File transfer is the dominant data transfer mode today and is expected to remain so in the future. GridFTP is widely used by the scientific communities to move files, with thousands of GridFTP servers deployed worldwide currently moving more than 1 PB per day. Clearly, it is important not only to understand the characteristics of these transfers but also to be able to control and optimize them.

In this paper, we focus on the problem of bandwidth allocation at the level of a protocol such as GridFTP. Although the bandwidth allocation problem has been extensively studied as a (theoretical) routing problem [8], [14], [25], [13], [18], wide-area transfer protocols today (and in the conceivable future) do not have any control over routers. Only limited work has been done in controlling bandwidth at the level of such a protocol. Managed GridFTP effort [7] included Globus Fork (GFork), a user-configurable super-server daemon similar to xinetd [22], which enables sharing of state across client connections and could effectively throttle the transfer rates for GridFTP client requests to prevent system meltdown. While the focus of the earlier work was on protecting the GridFTP resources from any unintentional or intentional misuse, here we focus on modeling the GridFTP throughput and using the models to control bandwidth allocation to various transfer requests.

Controlling bandwidth allocation for large-scale wide-area data transfers can be desirable for many reasons. First, it may be important to prioritize certain transfers over others. For example, consider a user who uses one of the XSEDE (a project for advanced cyberinfrastructure and digital services, funded by the National Science Foundation) [23] resources for a simulation and another XSEDE resource for analysis of the data. The transfer of data needs to be completed prior to the availability of the second computing resource to the

user; otherwise, cycles may be wasted. Second, given a site that is an endpoint for a large number of transfers, it will be desirable to schedule the transfers to meet certain objectives, such as minimizing the average wait time. Performing such scheduling will require some control over the resource (i.e., the bandwidth).

In controlling the bandwidth allocation, we first explore how the performance of individual GridFTP transfers can be controlled by varying the degree of *parallelism* (number of sockets used for a single transfer) and *concurrency* (number of transfers that are allowed to proceed at once). We find that concurrency is a more powerful control knob than is parallelism for data transfer rates. Next, we consider an end-point (source) whose available bandwidth for transfers is consumed by a small set of other endpoints (destinations). This is often the case in practice, since only certain major resources have large bandwidths, and similarly users performing large-scale science are more inclined to keep data transfers within these major resources. For controlling bandwidth allocation within these resources, we build a model for each destination's achieved throughput in terms of its concurrency and an indicator of the known source load. We then enhance this model by including an indicator of the time-varying external load. We propose several distinct ways to characterize and compute this external load.

Focusing on transfers within the five sites that are part of the XSEDE project, we study the effectiveness of the proposed models in controlling the bandwidth allocation. One of the best-performing models achieved a 15.6% mean error rate and a 10.4% median error during testing; 63% of the time this model's errors were below 15%, and 49% of the time they were below 10%.

A key strength of our approach is that we control bandwidth allocations on the client side using existing GridFTP options. No changes to the GridFTP protocol or to the deployed GridFTP servers are required. The alternative approach of changing the GridFTP protocol and getting many GridFTP implementors and sites to update their GridFTP implementations is a complex and time-consuming process. Our solution can be easily deployed on a system such as Globus Transfer [3], a GridFTP client that is being offered as a service to users.

The rest of this paper is organized as follows. In Section II, we provide background on GridFTP and describe the motivation for this study. Section III describes the key performance optimization techniques in GridFTP—parallelism and concurrency—and discusses the experiments to characterize them. In Section IV, we describe the different models we have built to predict the end-to-end throughput, and we discuss how we measure and capture the time-varying load in our experiments. Experimental results to evaluate the models' effectiveness in controlling the throughput for various destinations are presented in Section V. We discuss related work in Section VI, talk about deployment considerations in Section VII, and conclude in Section VIII.

II. Background and Motivation

This section reviews key details of GridFTP. GridFTP [2], [1] extends the legacy File Transfer Protocol (FTP) [21] to enable secure, reliable, and fast transport of bulk data. The GridFTP protocol has been implemented by Globus [12] and others and is widely used by scientific communities for bulk data movement. Globus Transfer [11], [3] is a centrally managed and hosted service for orchestrating data transfer tasks that involve file movement or synchronization between GridFTP servers. Globus greatly simplifies configuration and management for users, providing simple Web 2.0 interfaces and automating transfer management tasks.

The following are some of the use cases of GridFTP transfers:

- Transfer the output of a big simulation at one computing facility to another computing facility for analysis.
- Move the data generated at an experimental facility (such as DOE light sources, LHC) to a computing facility for analysis.
- Replicate simulation, experimental, or analysis data at various locations for distribution to users.
- Move data from a compute or experimental facility to a compute cluster at a user's home institution, desktop, or laptop, or even to a cloud provider, for further analysis.

Globus GridFTP servers support usage statistics collection. At the end of each transfer, GridFTP servers send the following information as a UDP packet to a usage collector run by Globus: transfer type (store or retrieve), size in bytes, start time of the transfer, transfer duration, IP address of the GridFTP server, number of parallel TCP streams, TCP buffer size, and block size.

To understand some of the trends associated with wide-area data transfers, we looked at the GridFTP usage logs for a 24-hour period for the top 10 sites that transferred the most number of bytes in that period. Results from three sites, namely, the first, fifth, and tenth most heavily loaded sites, are shown in Figures 1 and 2. The time axis is based on Chicago local time, but the GridFTP servers for which the usage is plotted could be in a different time zone. The transfer patterns show a large variability—sometimes there are no transfers, and sometimes there are many concurrent transfers. This situation is not surprising considering the typical activity pattern of most system administrators and end users. This points to the fact that the overall load can vary substantially over time. Furthermore, closer examination of these trends show that while there is a substantial variation over the 24-hour period, there is more stability over a shorter period. More specifically, in Figure 3, we show the variance over the entire 24-hour period, the four disjoint 6-hour periods, the 24 disjoint 1-hour periods, and each disjoint 15-minute period. We can see that the variance drops significantly for shorter durations. Thus, one can measure the performance of data transfers at a certain time and get a good indicator of the load for the immediate future.

III. Parallelism and Concurrency

Parallelism and *concurrency* are two key performance optimization mechanisms for GridFTP transfers. Parallelism involves the use of multiple socket connections to transfer chunks of a file in parallel from a single-source GridFTP server process to a single-destination GridFTP server process. Concurrency is the use of multiple GridFTP server processes at the source and destination. In the most common use of concurrency, each GridFTP server process transfers a different file. However, since GridFTP supports partial file transfers

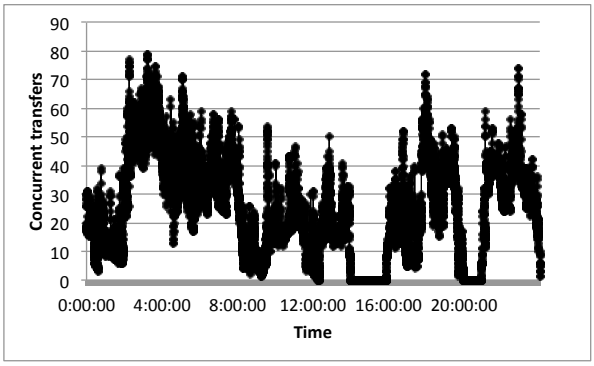


Fig. 1: Number of concurrent transfers over a 24-hour period for the most heavily used GridFTP server

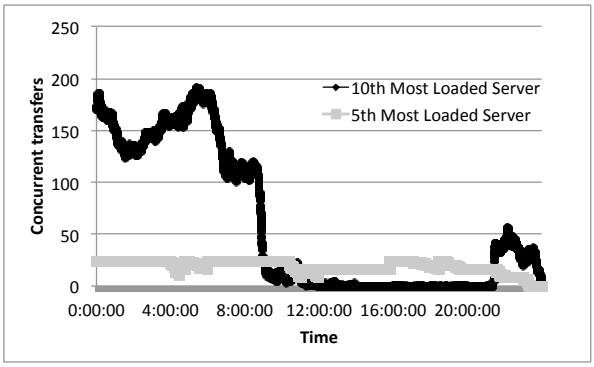


Fig. 2: Number of concurrent transfers over a 24-hour period for the 5th and 10th most loaded GridFTP servers

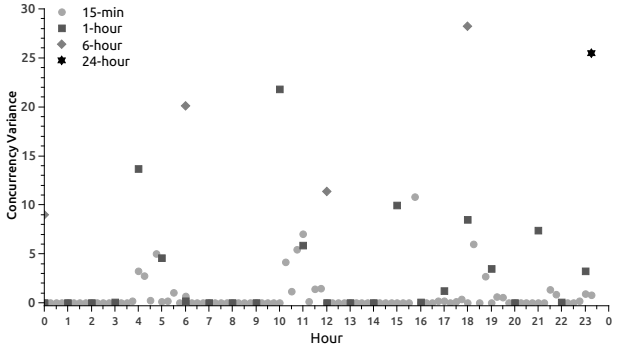


Fig. 3: Load variance over a 24-hour period for the 5th most loaded GridFTP server

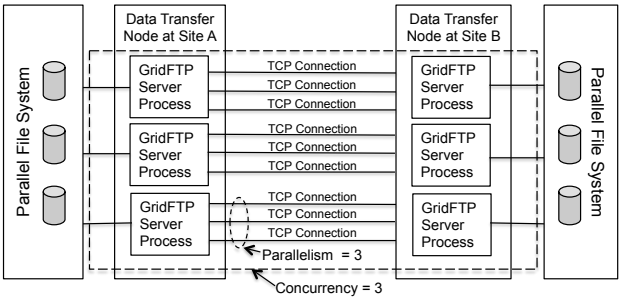


Fig. 4: Parallelism and concurrency in GridFTP

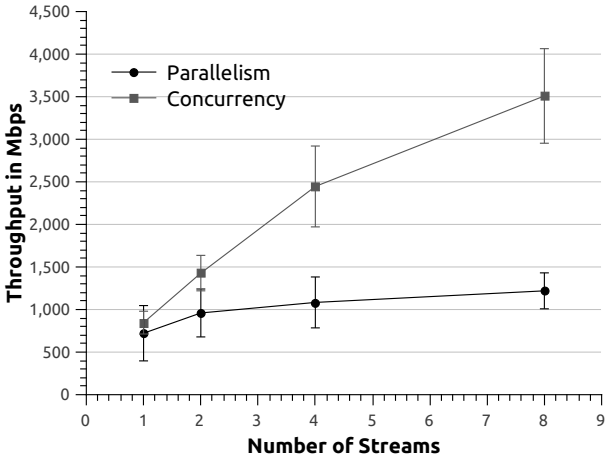


Fig. 5: Concurrency vs parallelism. Source: NERSC; Destination: Ranger

(transfer "X" bytes of data from offset "Y"), it is also possible, in principle, to have each of several GridFTP server processes transfer different portions of the same file. Figure 4 shows the difference between parallelism and concurrency in GridFTP.

Independent transfer requests to a GridFTP server normally lead to concurrent transfers, one for each request. The independent transfer requests could come from different users or from the same user. Even for a single transfer request, a GridFTP client can initiate concurrent transfers of multiple files, in the case of a *directory transfer* or *multifile transfer* request, or when performing concurrent transfers of chunks of a file, in the case of a *single file transfer* request. The latter scenario is similar to the use of parallelism except that concurrency involves multiple processes whereas parallelism involves multiple connections. We experimented with both concurrency and parallelism in order to understand how each one affects the performance of transfers.

Most large-scale data transfers can be expected to be between supercomputing facilities, since they are the ones with the ability to both store and process large amounts of data. For example, the usage statistics on data transfers in a recent XSEDE quarterly report [24] indicate that about 75% of the data transferred are between big compute facilities. Thus, the experiments we report in this paper are from a data transfer node (DTN) [10] at Stampede or Ranger, supercomputers at Texas Advanced Computing Center (TACC), or from a DTN at the National Energy Research Scientific Computing Center (NERSC), to one or more DTNs at Blacklight, a compute cluster at the Pittsburgh Supercomputing Center; Yellowstone, a supercomputer at the National Center for Atmospheric Research (NCAR); Kraken, a supercomputer at the University of Tennessee; and Gordon, a compute cluster at the San Diego Supercomputer Center (SDSC). These DTNs are dedicated for data transfer purposes. Since 10 Gbps wide-area network (WAN) connectivity is the norm at the big supercomputing facilities, we wanted the source DTN to have a 10 Gbps connectivity and also deliver aggregate disk-to-disk throughput as close to 10 Gbps as possible. The DTNs at Ranger and NERSC were able to deliver an aggregate throughput as high as 8 Gbps for disk-to-disk wide-area transfers, and the DTN at Stampede was able to deliver higher than 9 Gbps. We also

wanted to use a mix of DTNs with varying capabilities in terms of achievable throughput as destinations for the data transfers in our experiments. Yellowstone and Gordon were able to achieve a maximum throughput of around 5 Gbps for one-to-one transfers from Stampede. Blacklight achieved around 4 Gbps, and Mason and Kraken achieved around 2.5 Gbps. Unless otherwise noted, the experiment results reported are an average of at least five runs.

For concurrency experiments, in order to maintain the concurrency constant throughout a run, all transfers were timed to run for 90 seconds. File sizes used were at least 30 GB (big enough that even the most powerful destinations used in the experiments will take more than 90 seconds to transfer this file). As soon as all the transfers were started by an automated script, a procedure for tracking the number of concurrent transfers began to run. This procedure periodically checked whether exactly the intended number of concurrent transfers was running during the entire 90 seconds. It also checked each destination's log for errors; if it found any errors, it killed all transfers and exited from the script.

We show the results of a subset of the parallelism and concurrency experiments in Figures 5 and 6. Figure 5 compares the effect of parallelism and concurrency on throughput, whereas Figure 6 shows the effect of both parallelism and concurrency on throughput.

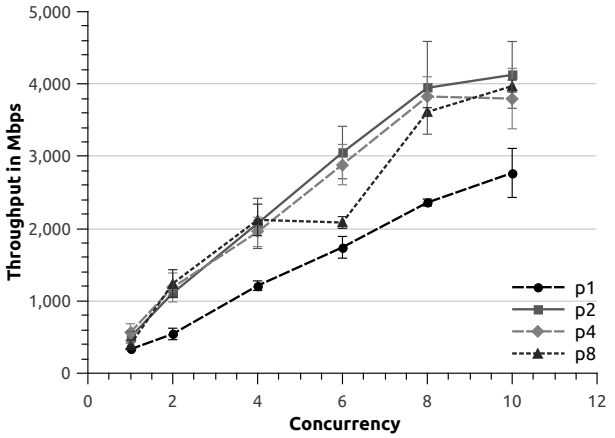


Fig. 6: Impact of concurrency and parallelism on throughput – Ranger to Blacklight

A key observation from our results is that concurrency turns out to be a more powerful control knob than is parallelism for increasing the throughput, as we can clearly observe from Figure 5. Moreover, we can see from Figure 6 that the throughput achieved for *concurrency* (cc) of x and *parallelism* (p) of y is greater than the throughput achieved for *concurrency* of y and *parallelism* of x (for any x greater than y). For example, the average throughput for $cc = 8$ and $p = 1$ is >2.2 Gbps, whereas the average throughput for $p = 8$ and $cc = 1$ is <600 Mbps. The likely reason for this behavior is as follows. As far as the network interface controller and the wide-area connections are concerned, parallelism and concurrency work in the same way. But there is an important distinction within the source or the destination. The multiple processes used when concurrency is increased seem to help get better I/O performance. We note that GridFTP uses the *asynchronous event model*, and use of parallelism implies additional asynchronous reads from the disk. However, it does not create multiple threads to read from the disk, a limitation that is resolved with the use of concurrency. We also note that concurrency does increase the net load on the server because of these processes, and it could imply a negative impact on the performance of other transfer jobs executing at these servers.

IV. Modeling and Adapting Destination Throughputs

In this section we present the model formulation, describe the different models we have developed, and discuss how to capture the time-varying load in our experiments.

A. Problem Formulation

Our objective is to control the allocation of bandwidth available for transfer(s) from a source to the destination(s). As we have already established, concurrency is a powerful knob for controlling bandwidth allocation.

We consider a situation where one endpoint is heavily loaded and most of its bandwidth is consumed by transfers occurring to or from a small number of other sites. Our example of such a situation involves the Stampede system, with associated transfers to or from other nodes that are part of the XSEDE project (Blacklight, Gordon, Kraken, Mason, and Yellowstone). This example is important to optimize for, because Stampede serves as a major data repository and computational resource, and data that is either generated or archived needs to be transferred to other sites, stressing the available bandwidth. The example is also realistic in that most of the associated bandwidth is consumed by transfers to a small number of other sites. The reason is twofold: (1) only a small number of sites have the large bandwidth at their end, and thus transfers to or from them are the only ones that can consume a substantial fraction of bandwidth at an endpoint like Stampede; and (2) only a small number of sites have computational resources to store and analyze large-scale data. While one can expect a number of concurrent transfers to numerous other sites, they are likely to involve smaller files and/or will each consume only a very small amount of bandwidth.

In such a scenario, our goal is to develop a simple model for GridFTP transfer throughput based on a small number of important parameters. Particularly, it appears that the bandwidth available for transfers between the heavily loaded end-point A and one of its key destinations B will depend on the following factors:

- The total number of ongoing transfers (concurrency) between the endpoint A and all its *major* transfer endpoints, such as the five other resources that are part of the XSEDE project in the case of Stampede. The larger the number of ongoing transfers, the further the bandwidth available to B is going to be split, since the outgoing bandwidth from A is the major constraint.
- The total number of ongoing transfers (concurrency) between the endpoint A and the endpoint B. We can expect that the higher the concurrency to the endpoint B, the higher the bandwidth to B will be.
- All other *load* in the system—which can include transfers to other sites (with less bandwidth); other activity on the

endpoints, such as disk accesses; and other load on the wide-area network, for example, transfers to and from various other end-points in the entire network.

For simplicity, we refer to the first item as the *source concurrency* and the second item as the *destination concurrency*, although transfers can occur in either direction. The last item is referred to as *external load* in our discussion. The first two items are known to the GridFTP or a similar service; furthermore, we are assuming that both can be controlled (varied) by the service. In comparison, the last item, external load, cannot be controlled and is also hard to measure.

B. Using Source and Destination Concurrency to Predict Throughputs

Our first step was to see whether models that consider only source and destination concurrency for controlling the destination bandwidth are adequate. We initially focused on linear models that relate destination throughput to destination concurrency and source concurrency. Recall that a linear model between several input variables (or features) $X_1, X_2, \ldots, X_k$ and a target variable Y is

$$Y^{'} = a_1X_1 + a_2X_2 + \ldots + a_kX_k + b,$$

where $Y^{'}$ is the prediction of the observed value of Y for the corresponding values of X_i.

Specifically, we trained the following two linear models:
1. destination throughput (DT) as the target variable and *destination concurrency* (DC) and *source concurrency* (SC) as input variables.

$$DT = a_1 \times DC + a_2 \times SC + b_1 \quad (1)$$

2. Destination throughput (DT) as the target variable and the ratio between the *destination concurrency* and the *source concurrency* (DC/SC) as a basis function.

$$DT = a_3 \times \frac{DC}{SC} + b_2 \quad (2)$$

To create training and validation sets, we performed load variation experiments to characterize the performance of a destination in terms of the source and destination concurrency. In these experiments, we start with a baseline case (a case in which all of the destinations have the same concurrency), we continue by increasing destination concurrency for a destination by 1, we run the baseline case again, we increase destination concurrency for the next destination by 1, we run the baseline again, and so on. Parallelism was held constant at eight streams for all destinations and for the entire experiment. A separate model was built for each destination, since each destination has different characteristics.

We used three-fifths of the data from our experiments for training and two-fifths of the data for validation. In other words, for each experiment, for each destination, a fraction of the data from that destination went to training and the rest to validation. After a model is built by using the training data, it will not fit the training data perfectly. The resulting error rate is called the *training error*. The model is then validated by using the validation data; the resulting error rate is called the *validation error*. The coefficients of the models are calculated by using the regularized least squares method [6]. The regularizer, λ, was sampled from $\{0, 100\}$ on a logarithmic scale (47 regularizers total, sampled more sparsely as the value of λ increases). From these potential models, we picked one that produced $\min_\lambda(TE+VE)$, where TE and VE are the training and validation errors, respectively.

TABLE I: Training and validation errors (in percentages) for log and non-log models without external load.

Destination	Nonlog Model		Log Model	
	Training Error	Validation Error	Training Error	Validation Error
Blacklight	17.26	17.36	14.04	14.79
Gordon	11.99	9.69	8.78	9.57
Kraken	17.31	16.60	16.20	15.75
Mason	15.53	13.56	14.94	13.24
Yellowstone	12.02	13.68	8.97	10.57

The training and validation errors were comparable to each other and higher than 15% in almost all cases. We therefore considered other models. Particularly, our hypothesis was that some of the nonlinear dependencies (such as throughput saturation) between the terms could be captured through a model in the form of

$$Y^{'} = X_1^{a_1} \times X_2^{a_2} \times \ldots \times X_k^{a_k} \times 2^b. \quad (3)$$

After experimentation with a number of models, the following turned out to be effective. (Unless otherwise noted, the term "log" in the rest of the paper refers to "log base 2.") Again, using DT, DC, and SC to denote destination throughput, destination concurrency, and source concurrency, we have

$$\log(DT) = a_4 \times \log(SC) + a_5 \times \log(DC) + b_3. \quad (4)$$

Note that solving Equation 4 for DT will yield an expression in the form of Equation 3:

$$DT = SC^{a_4} \times DC^{a_5} \times 2^{b_3}. \quad (5)$$

To show the effectiveness of this model, we present training and validation errors for the model captured through Equation 4, and we compare this model with the model captured through Equation 1 (best among the non-log models) in Table I. We can see that the log-based model is clearly better: the training and validation errors went down in every single case and up to 27%. Still the relative error rate is around 15% in many cases.

C. Incorporating External Load in the Model

The discussion has shown that a log model can be better than linear models, but it still has high errors. Thus, a model based on just the source load and destination concurrency clearly is too simplistic, and we need to characterize external load and incorporate it in our model. This situation, however, raises two questions: (1) How do we measure the external load impacting the transfers at any given time? and (2) How do we include the external load in our model(s)? Recall that we view all factors outside the source concurrency and destination concurrency as the external load, which can include network, disk, and CPU activities outside the transfers within the major resources we are modeling.

Our approach to factoring load in our model was as follows. We had observed that transfer activities tend to be more stable over a shorter duration of time but can vary significantly over the entire day. Thus, we obtained training data on different days and at different times of the day, and we had multiple data points that involved the same destination and source concurrency. The difference in the achieved throughput across runs with the same source and destination concurrencies can be attributed to the different external load.

Based on this reasoning, we considered the following three different basis functions for the external load (EL). The first function, $EL1$, was calculated as follows. First, for each transfer t in the experimental data, we found all transfers in both sets that had the same destination and source concurrency as t. Next, we calculated the average throughput AT for this subset of transfers. Then,

$$EL1 \ = \ T \ - \ AT, \tag{6}$$

where T is the observed throughput of transfer t. Intuitively, a negative external load (EL) helps account for a lower-than-average observed throughput. The second basis function, $EL2$, differs from $EL1$ in that instead of using the average throughput AT of transfers with matching DC and SC, we use the transfers' maximum throughput, MT:

$$EL2 \ = \ T \ - \ MT \tag{7}$$

Both functions are assuming that the external load has an additive impact. In practice, however, the load could have a multiplicative impact on the achieved throughput. To capture this possibility, we introduce the third function, $EL3$, calculated as follows.

$$EL3 \ = \ T/MT \tag{8}$$

To capture the measured load, we considered the following models.

$$DT \ = \ a_6 \times DC \ + \ a_7 \times SC \ + \ a_8 \times EL + b_4 \tag{9}$$

$$DT = SC^{a_9} \times DC^{a_{10}} \times AEL\{a_{11}\} \times 2^{b_5} \tag{10}$$

Equations 9 and 10 represent a non-log model and a log model, respectively, each with an external load feature. For Equation 10, the $AEL\{a_{11}\}$ feature, or adjusted external load, is defined as follows.

$$AEL\{a_{11}\} = \begin{cases} EL^{a_{11}} \text{ if } EL > 0 \\ |EL|^{(-a_{11})} \text{ otherwise} \end{cases} \tag{11}$$

This is done in order to avoid complex numbers in the calculations. Note that Equation 10 has the added benefit that, if necessary, it can model the sublinear growth of its features more effectively than a non-log model can. Hence, Equation 10 has the potential of being more effective for modeling saturation regions.

Table II shows the validation errors for both non-log and log models with external load as one of the features. We can see that these models are clearly better: validation errors went down by up to 45%. Note that relative error rates for all destinations are less than 10% with the log + $EL3$ model and less than or equal to 15% for all the models. Note also that in almost all cases the $EL2$ method does not work as well as the $EL1$ and $EL3$ methods. The likely reason is that deviation from average is less subject to random variation than is deviation from the maximum; moreover, load has more of a multiplicative impact, as opposed to the additive impact.

TABLE II: Validation errors (in percentages) for log and non-log models with differing definitions of external load feature. $EL1$, $EL2$, and $EL3$ are defined in Equations 6, 7, and 8.

Destination	Non-Log Model			Log Model		
	EL1	EL2	EL3	EL1	EL2	EL3
Blacklight	14.14	15.32	12.46	9.76	12.73	8.80
Gordon	5.31	9.08	8.31	6.73	8.73	7.70
Kraken	12.04	13.37	10.67	11.98	13.34	9.33
Mason	6.44	12.93	12.92	13.68	13.40	9.94
Yellowstone	9.21	11.41	10.22	8.11	9.41	8.23

D. Calculating External Load While Adapting Throughput

We have developed a model to express the throughput in terms of destination concurrency, source concurrency, and external load. As opposed to destination and source concurrency, which are known values, external load is unknown. The method we have used for obtaining load values for training models involves having multiple data points with the same source and destination concurrency values. When we are trying to adapt rate of transfers to one or more destinations, we cannot expect to have recent transfers with all possible combinations of source and destination concurrency values. At the same time, we can expect to have available data from transfers that have taken place recently. Similarly, we can expect that the external load does not change substantially over a few minutes.

Overall, we propose three ways to compute the external load (EL) for a given transfer:

1. Previous Transfer (PT) – uses the most recent previous transfer's EL (PEL), to compute EL for the current transfer (CEL). This can be done by solving the destination's model for EL (e.g., $EL = (DT - a_6 \times DC - a_7 \times SC - b_4)/a_8$ for the model represented by Equation 9) and plugging in the previous transfer's DC_{prev}, SC_{prev}, and observed throughput to obtain PEL. CEL is then defined as $PEL \times DC_{cur}/DC_{prev}$, where DC_{cur} is the current transfer's planned destination concurrency.

2. Recent Transfer (RT) – uses all the transfers in the past 30 minutes to the given destination to compute EL for the current transfer. This involves computing EL for each transfer to the given destination in the past 30 minutes, using destination and source concurrency used and the observed throughput, finding the average of these ELs, and using it as the EL for current transfer.

3. Recent Transfers with Error Correction ($RTEC$) – use all the transfers in the past 30 minutes to the given destination with error correction based on historical data to compute EL for the current transfer. For each transfer to the given destination in the past 30 minutes, one does the following:

- Calculate the average error in the model for (historical) datapoints with matching destination and source concurrency.
- Adjust observed throughput based on the error as calculated above.

- Compute EL using destination and source concurrency used and the adjusted observed throughput as computed above.

We then find the average of these ELs and use it as the EL for the current transfer.

As a next step, we generated new experimental data (with Stampede as source and Gordon, Mason, Yellowstone, Blacklight, and Kraken as destinations) and used various already-trained models in conjunction with these three methods of estimating external load to evaluate their effectiveness. Note that with different models, different methods for calculating load during training runs, and different methods for estimating the current load, we have a large number of combinations. Table III shows results from a set of heuristics that gave the best results. The cross-validation errors were uniformly low, indicating that all these methods are effective. There is no consistent trend between the three methods for estimating the current load, but the differences in results are also small.

Table IV shows how the log models with $EL1$ look for all the destinations. Mason's data transfer node was weakest among all the destinations used here, and it was able to achieve a throughput close to its maximum achievable throughput even with a low destination concurrency value. The low exponent for DC in the model for Mason reflects this characteristic. Gordon and Yellowstone are the strongest among all the destinations. Thus, incremental gain per destination concurrency is high for them. At the same time, as the load on the source increases, these destinations lose more throughput as well because the relative loss of cycles at the source for the powerful destinations is higher. Higher exponents for DC and SC in the model capture this behavior. Blacklight and Kraken have more intermediate strengths than do the other nodes; but Kraken is more similar to Mason, and Blacklight is more similar to Gordon. These results are also reflected in the weights for DC and SC in their models. Even though the exponents for AELs in the models shown in Table IV are low, we note that AEL is still a significant component since its value is an order of magnitude or more higher than DC and SC.

In an additional effort to understand how each feature in our models affects the error rate, we performed an ablation study. We picked the following models for the study: log with $EL1$ using $RTEC$ (a relatively complex model), the log model shown in Table I (because it is our best-performing model without the EL feature), and three new models—one using only DC to predict throughputs, one using only SC, and an overly simplistic model that has no inputs and always outputs the mean of the throughputs from the training set (we use this as a baseline model). A comparison of the simple models' performance with the other, more complex models helps put the errors in perspective. The findings from the ablation study are as follows. The mean testing error for the most complex model is 14.7%. This is a 12% improvement compared with that of the log model without EL, an 18-20% improvement compared with using the models with only SC or DC, and a 47% improvement compared with the baseline model. These results clearly show that the proposed models with the external load feature are much more effective in predicting the throughputs than are the other, less-complex models.

TABLE III: Cross-validation errors (in percentages): comparison of EL estimation methods for best-performing models with external load (based on validation errors).

Destination	Log Model w/ EL1			Log Model w/ EL3		
	PT	RT	RTEC	PT	RT	RTEC
Blacklight	18.78	19.54	20.52	21.54	18.68	20.68
Gordon	12.51	9.14	9.65	13.36	9.18	10.06
Mason	16.77	13.31	14.07	18.48	18.64	20.70
Kraken	21.47	17.23	16.83	22.01	18.39	18.01
Yellowstone	12.24	12.40	12.66	12.38	12.29	12.46

TABLE IV: Log models with $AEL1$ as external load. DT is throughput in Mbps. $AEL1$ is in Mbps and follows the definition in Equation 11.

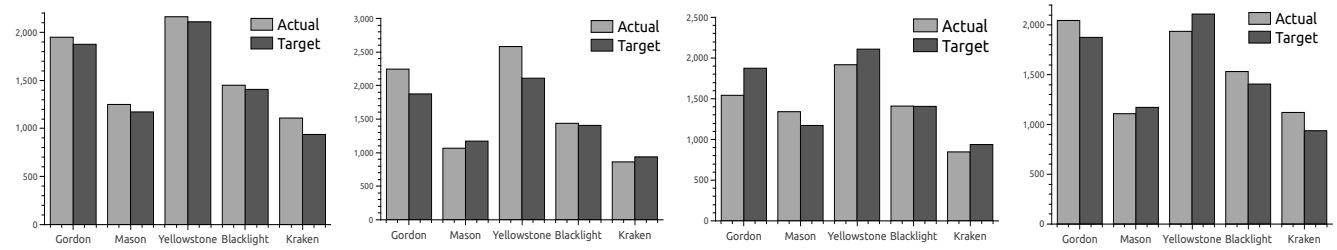
Destination	Model
Blacklight	$DT = DC^{0.868} \times AEL1\{0.028\} \times 2^{9.996}/SC^{0.283}$
Gordon	$DT = DC^{0.823} \times AEL1\{0.019\} \times 2^{13.221}/SC^{0.877}$
Kraken	$DT = DC^{0.437} \times AEL1\{0.031\} \times 2^{10.738}/SC^{0.234}$
Mason	$DT = DC^{0.080} \times AEL1\{0.038\} \times 2^{10.697}/SC^{0.171}$
Yellowstone	$DT = DC^{0.788} \times AEL1\{0.015\} \times 2^{12.884}/SC^{0.724}$

V. Applying Models to Control Transfer Bandwidths

Recall that our goal was to be able to control the allocation of bandwidth to different destinations when the source is already saturated. To this end, we have developed models that can relate source and destination concurrency levels to the throughput obtained by each destination. Applying these models for a particular transfer also involves certain challenges, and we now show how we approach the problem.

Earlier, we described three effective methods of estimating external load for a new transfer, which means that now we have efficient ways of calculating EL. Given a target throughput for a destination, we would like to determine the DC that would get us closest to the target. However, there is often more than just one destination to transfer data to or from the source, which means that SC is also unknown. To narrow the search space, we can limit the maximum concurrency value for each destination to a value such as 20, which is reasonable because some of the destinations had a cap of 30 for all concurrent transfers.

However, even a concurrency limit of 20 leads to a large number of possible combinations of destination concurrencies (20^n, for n destinations). Rather than doing an exhaustive search, we use an optimized method, shown in Algorithm 1. If each destination has a concurrency between 1 and 20 (inclusive), the number of possible distinct source concurrency values is $19 \times ND + 1$ (or $SC_{max} - ND + 1$), where ND is the number of destinations. Therefore, we compute ELs for all destinations using their respective models, similarly compute the DC value for each destination, and pick the permutation that satisfies the following: (1) the sum of concurrency values of the destinations does not deviate too much from the source concurrency value, and (2) each destination's error function, defined as the difference between target destination throughput (TDT) and predicted destination throughput (PDT), is



(a) (b) (c) (d)

Fig. 7: Results of a ratio experiment. Throughputs are shown in Mbps. Ratios are 4:5:6:8:9 for Kraken, Mason, Blacklight, Gordon, and Yellowstone. (a) Concurrencies picked by Algorithm 1 were {1,3,3,1,1}. Model: log with $EL1$. Method: $RTEC$. (b) Concurrencies picked by Algorithm 1 were {1,4,3,2,2}. Model: log with $EL3$. Method: $RTEC$. (c) Concurrencies picked by Algorithm 1 were {1,1,3,1,1}. Model: log with $EL1$. Method: RT. (d) Concurrencies picked by Algorithm 1 were {1,4,3,1,1}. Model: log with $EL3$. Method: RT.

minimized. This process is shown in Algorithm 1. Note that although the PT method depends on the new transfer's DC for its external load estimate, one can still solve for and isolate DC in the models.

Algorithm 1 Obtaining concurrency configurations for given target throughputs using a narrowed search space

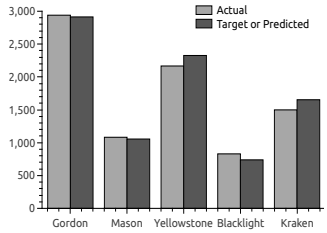
function GETCCS()
  $SC_{max} \leftarrow 100$
  δ is a user-defined variable
  $E_i \leftarrow \infty, i \in \{1, ND\}$ ▷ Error for each destination
  for $SC = n \rightarrow SC_{max}$ **do**
    for $i = 1 \rightarrow n$ **do**
      $DC_i \leftarrow \sqrt[a]{\frac{TDT_i}{SC^b \times EL^c \times 2^d}}$
      $DC_i \leftarrow round(DC_i)$ ▷ $Pr(DC_i) \notin \mathbb{Z}$ is high
      $PDT_i \leftarrow DC_i^a \times SC^b \times EL^c \times 2^d$
    end for
    $E'_i \leftarrow |PDT_i - TDT_i|, i \in \{1, ND\}$
    if $(E'_i < E_i \ \forall \ i) \wedge (|\sum_{i=1}^{ND} DC_i - SC| < \delta)$ **then**
      $CC \leftarrow [DC_1 \ \ldots \ DC_n]$
      $E_i \leftarrow E'_i, \ \forall \ i$
    end if
  end for
  return CC
end function

We performed two types of experiments: *ratio* experiments, which allocate the available bandwidth at the source among the destinations based on a predefined ratio, and *factoring* experiments, where the goal is to increase the throughput of a particular destination by a certain factor over the base rate it is obtaining when the source is saturated.

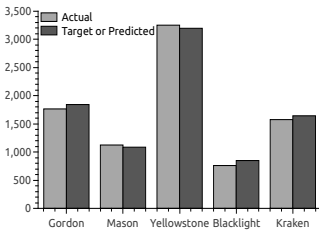
For the *ratio* experiments, the goal is to achieve a specific fraction of bandwidth for each destination. We experimented with the ratios 2:1:2:3:3, 3:2:3:4:4, 4:5:6:8:9, and 5:4:5:8:9 for Kraken, Mason, Blacklight, Gordon, and Yellowstone. These combinations were picked based on the maximum throughputs that can be independently achieved by these destinations in various tests.

The *factoring* experiments represent the case when we need to increase the bandwidth to one particular destination because of certain priorities or scheduling needs. Specifically, let us say that there is a base case (in our experiments, each destination had a concurrency of 2 and parallelism of 8 to move data from the source.) We used the throughput obtained by each destination in this experiment as the baseline, and we want to increase the throughput for each destination (separately) to 1.5 times (and 2 times) the baseline throughput. Algorithm 1 was used to determine a concurrency value to achieve such a target throughput. Note that for each of these experiments, the search space is five times smaller than for the ratio experiments because only one destination concurrency is unknown at any time: that of the destination whose throughput we are attempting to increase.

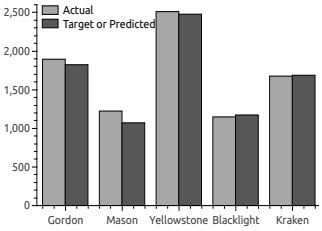
Figure 7 shows the results of the ratio experiment for the top four models/methods (log $EL1$/$EL3$ models and RT/$RTEC$ methods) in terms of the cross validation error that was shown earlier in Table III. We can see that all four model/method combinations are effective in predicting the throughputs, although log $EL1$ with RT is not as good as the other three. Log $EL1$ with $RTEC$ is the best among all four; errors for four of the destinations were less than 5%, and for the other destination it was 10.5%. $RTEC$ method performs better than RT, as expected, because $RTEC$ uses both the most recent transfers and the historical data (for error correction) to calculate the external load. For the remaining experiments, we show the results for only log $EL1$ with $RTEC$; the results of the other three approaches were comparable to this approach. Figure 8 shows the results of a factoring experiment where we attempt to increase each destination's throughput to 1.5 times (and then to 2 times) its baseline and try to predict the other destinations' throughputs as their concurrencies are held constant. The figure shows the results for Gordon, Yellowstone, Blacklight, Kraken, and Mason. For all experiments, Algorithm 1 (with reduced search space as mentioned in the above paragraph) was used to obtain destination concurrencies. The combined results of 1.5x and 2x factoring experiments presented in Figure 8 can be summarized as follows: 83.6% of the errors are below 15%, and 65.5% of them are below 10%. The ratio and factoring experiments were performed three times, and the overall results (including all four ratio combinations and the two factoring experiment multipliers) are as follows for the log model with $EL1$ and $RTEC$: a mean error of 15.6%, with 63% of the errors below 15% and 49% of them below 10%. Overall, we see that these models



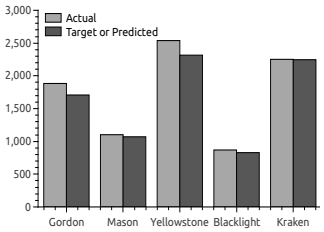
(a) Increasing Gordon's baseline throughput by 1.5x. Concurrency picked by Algorithm 1 for Gordon was 4.



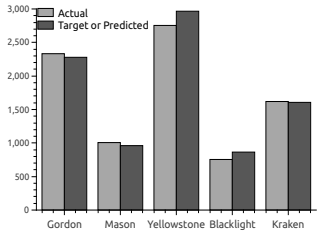
(b) Increasing Yellowstone's baseline throughput by 1.5x. Concurrency picked by Algorithm 1 for Yellowstone was 3.



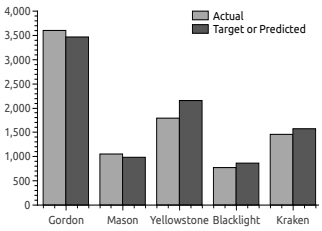
(c) Increasing Blacklight's baseline throughput by 1.5x. Concurrency picked by Algorithm 1 for Blacklight was 3.



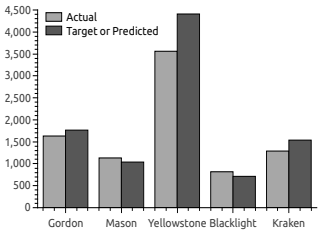
(d) Increasing Kraken's baseline throughput by 1.5x. Concurrency picked by Algorithm 1 for Kraken was 4.



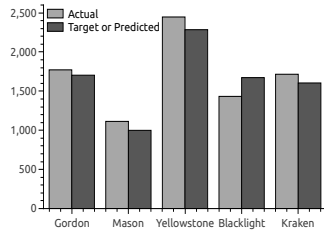
(e) Increasing Mason's baseline throughput by 1.5x. Concurrency picked by Algorithm 1 for Mason was 1.



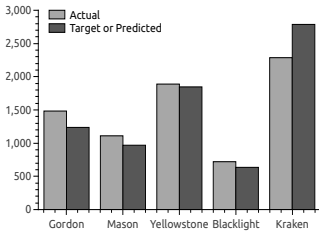
(f) Increasing Gordon's baseline throughput by 2x. Concurrency picked by Algorithm 1 for Gordon was 5.



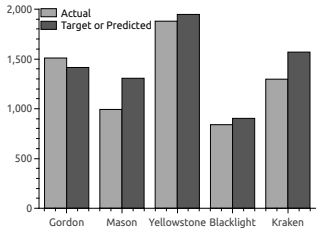
(g) Increasing Yellowstone's baseline throughput by 2x. Concurrency picked by Algorithm 1 for Yellowstone was 4.



(h) Increasing Blacklight's baseline throughput by 2x. Concurrency picked by Algorithm 1 for Blacklight was 4.



(i) Increasing Kraken's baseline throughput by 2x. Concurrency picked by Algorithm 1 for Kraken was 10.



(j) Increasing Mason's baseline throughput by 2x. Concurrency picked by Algorithm 1 for Mason was 7.

Fig. 8: Results of a factoring experiment. Y-axis is throughput in Mbps. Attempting to increase each destination's throughput to 1.5 times and 2 times its baseline throughput. Other destinations' concurrencies are kept constant as we attempt to predict their throughputs. Model: log with $EL1$. Method: $RTEC$.

are effective in controlling bandwidth allocation.

VI. Related Work

Several models have been proposed for predicting the behavior of parallel TCP streams and finding the optimal number of streams [9], [17], [15], [19], [4]. Some of these models (for example, [15]) work only for uncongested networks, and these models were evaluated mostly by using simulations (i.e., not with real data transfer applications). Several studies [20], [16], [26] have developed models to find the optimal number of TCP streams to use for data transfer in the context of GridFTP. Among these, Yildirim *et al.* [26] improves upon the earlier models to find the optimal number of TCP streams to use for a transfer with minimal history information and low prediction overhead. Ohsaki and Imase [20] propose a model based on fluid-flow approximation to find both the optimal number of TCP streams and the optimal TCP buffer size to use for a transfer. Since then, the autotuning mechanism in Linux and other operating systems has improved significantly, to the point that there is no need for any TCP buffer size tuning in the application. The major difference between those studies and our work is that they focus on optimal number of streams to use to get the maximum throughput on the network. In comparison, we attempt to model the GridFTP transfer throughput based on the end-to-end performance behavior, taking into account the load on the end-system, variabilities in destinations' capabilities, and the number of concurrent transfers. Our emphasis has also been on concurrency, and we have shown that it is a more powerful knob to control the end-to-end throughput than are the parallel TCP streams.

Many studies on bandwidth allocation have been conducted [8], [14], [25], [13], [18]. Kumar and Kleinberg [18] propose an iterative algorithm for computing a max-min fair set of allocations. Goel *et al.* [13] describe an algorithm that approximates max-min fairness as well as optimizes the throughput. These efforts have been at the level of router implementation, whereas our focus has been on end-application-level control.

VII. Discussion

Three common GridFTP deployments exist:

- Standard (nonstriped) GridFTP server on one or more DTNs with DNS round robin for load balancing (in case of more than one DTN).
- Standard GridFTP server on one or more DTNs, associating all servers with a single logical endpoint on Globus transfer [11], relying on Globus for load balancing.
- Striped GridFTP server [1], with one head node and multiple data mover nodes, where each data mover is responsible for moving a portion of a file.

Our current approach will work well in all three scenarios, and it does not require any changes to the GridFTP protocol or the GridFTP server code. An implementation of the pseudocode on page 8 on the GridFTP client would suffice. It can be easily deployed on a system such as the Globus transfer service, a GridFTP client that is being offered as a service to users.

We note that models presented in this paper have been trained using data obtained from experiments that were run on production environments. Hence, all the participating nodes (both for the experiments that produced the training data and for the experiments that tested the models' effectiveness) simultaneously acted as both destinations and sources while we ran our tests, because of some transfers that were not within our control. The impact of those transfers on the throughput for our transfers has been captured by using the *external load* feature in our model. We also note that the models' training overheads are low and that the models can be quickly retrained, if needed.

VIII. Conclusions

This paper has focused on understanding performance issues with large-scale wide-area transfers and on controlling bandwidth allocation for large transfers at the level of a file transfer protocol. Our emphasis has been on transfers between several of the leadership-class machines at major supercomputing centers or laboratories, since they have not only high bandwidths between them but also a very high load.

After establishing that concurrency turns out to be a more powerful control knob than is parallelism for data transfer throughputs, we focused on developing models that can help us control bandwidth allocation. We have shown that log models that combine total source concurrency, destination concurrency, and a measure of external load are effective. Much of our focus has been on measuring and accounting for the external load, and we have shown that methods that utilize both recent and historical experimental data in estimating external load prove to be robust.

Overall, this work has accomplished two goals. First, we have developed models that capture how bandwidth is shared across different destinations. Second, we have provided a useful mechanism for controlling bandwidth allocation by changing concurrency levels to one or more destinations.

Acknowledgment

This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357, and by the National Science Foundation awards ACI-1339757 and ACI-1339798.

References

[1] B. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The Globus striped GridFTP framework and server. In *SC'2005*, 2005.

[2] W. Allcock. Gridftp: Protocol extensions to ftp for the grid, 2003.

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

[4] E. Altman and D. Barman. Parallel tcp sockets: Simple model, throughput and validation. In *In: IEEE INFOCOM 2006*, 2006.

[5] Biological and environmental research network requirements workshop, april 2010 - final report. http://www.es.net/assets/Papers-and-Publications/BER-Net-Req-Workshop-2010-Final-Report.pdf.

[6] C. M. Bishop and N. M. Nasrabadi. *Pattern Recognition and Machine Learning*, volume 1. Springer New York, 2006.

[7] J. Bresnahan, M. Link, R. Kettimuthu, and I. Foster. Managed gridftp. In *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW '11, pages 907–913, Washington, DC, USA, 2011. IEEE Computer Society.

[8] Z. Cao and E. W. Zegura. Utility max-min: An application-oriented bandwidth allocation scheme. In *INFOCOM*, pages 793–801, 1999.

[9] J. Crowcroft and P. Oechslin. Differentiated end-to-end internet services using a weighted proportional fair sharing tcp. *SIGCOMM Comput. Commun. Rev.*, 28(3):53–69, July 1998.

[10] Data Transfer Nodes (DTNs). http://fasterdata.es.net/science-dmz/DTN/.

[11] I. Foster. Globus Online: Accelerating and democratizing science through cloud-based services. *IEEE Internet Computing*, May:70–73, 2011.

[12] Globus. http://www.globus.org.

[13] A. Goel, A. Meyerson, and S. Plotkin. Combining fairness with throughput: online routing with multiple objectives. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, STOC '00, pages 670–679, New York, NY, USA, 2000. ACM.

[14] R. Guerin, H. Ahmadi, and M. Naghshineh. Equivalent capacity and its application to bandwidth allocation in high-speed networks. *IEEE J.Sel. A. Commun.*, 9(7):968–981, Sept. 2006.

[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 *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, IPDPS '02, pages 314–, Washington, DC, USA, 2002.

[16] T. Ito, H. Ohsaki, and M. Imase. Gridftp-apt: Automatic parallelism tuning mechanism for data transfer protocol gridftp. In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '06, pages 454–461, Washington, DC, USA, 2006. IEEE Computer Society.

[17] G. Kola and M. K. Vernon. Target bandwidth sharing using endhost measures. *Perform. Eval.*, 64(9-12):948–964, Oct. 2007.

[18] A. Kumar and J. Kleinberg. Fairness measures for resource allocation. *SIAM J. Comput.*, 36(3):657–680, Sept. 2006.

[19] D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante. Modeling and taming parallel tcp on the wide area network. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers - Volume 01*, IPDPS '05, pages 68.2–, Washington, DC, USA, 2005. IEEE Computer Society.

[20] H. Ohsaki and M. Imase. On modeling gridftp using fluid-flow approximation for high speed grid networking. In *Proceedings of the 2004 Symposium on Applications and the Internet-Workshops (SAINT 2004 Workshops)*, SAINT-W '04, pages 638–, Washington, DC, USA, 2004. IEEE Computer Society.

[21] J. Postel and J. Reynolds. File transfer protocol (ftp), RFC 959, 1985.

[22] xinetd, the extended internet services daemon. http://linux.die.net/man/8/xinetd.

[23] Extreme Science and Engineering Discovery Environment (XSEDE). https://www.xsede.org/.

[24] XSEDE Project 2013 First Quarter Report. https://www.ideals.illinois.edu/handle/2142/44872.

[25] H. Yaïche, R. R. Mazumdar, and C. Rosenberg. A game theoretic framework for bandwidth allocation and pricing in broadband networks. *IEEE/ACM Trans. Netw.*, 8(5):667–678, Oct. 2000.

[26] E. Yildirim, D. Yin, and T. Kosar. Prediction of optimal parallelism level in wide area data transfers. *IEEE Trans. Parallel Distrib. Syst.*, 22(12):2033–2045, Dec. 2011.