Data transfer between scientific facilities – bottleneck analysis, insights, and optimizations

Yuanlai Liu[§], Zhengchun Liu[*], Rajkumar Kettimuthu[*], Nageswara S.V. Rao[¶], Zizhong Chen[§] and Ian Foster[*†]
[*] Data Science and Learning Division, Argonne National Laboratory, 9700 Cass Ave., Lemont, IL 60439, USA
{zhengchun.liu, kettimut, foster}@anl.gov, raons@ornl.gov, chen@cs.ucr.edu, yliu158@ucr.edu
[†] Department of Computer Science, University of Chicago, Chicago, IL 60637, USA
[§] University of California, Riverside, 900 University Ave., Riverside, CA 92521, USA
[¶] Oak Ridge National Laboratory, 1 Bethel Valley Rd., Oak Ridge, TN 37830, USA

***Abstract*—Wide area file transfers play an important role in many science applications. File transfer tools typically deliver the highest performance for datasets with a small number of large files, but many science datasets consist of many small files. Thus it is important to understand the factors that contribute to the decrease in wide area data transfer performance for datasets with many small files. To this end, we (i) benchmark the performance of subsystems involved in end-to-end file transfer between two HPC facilities for a many-file dataset that is representative of production science transfers; (ii) characterize the per-file overhead introduced by different subsystems; (iii) identify potential dependencies and bottlenecks; (iv) study the effectiveness of transferring many files concurrently as a means of reducing per-file overheads; and (v) prototype a prefetching mechanism as an alternative of concurrency to reduce the per-file overhead on source storage system. We show that both concurrency and prefetching can help reduce the per-file overhead significantly. A reasonable level of concurrency combined with prefetching can bring the per-file overhead down to a negligible level.**

I. Introduction

Massive amounts of data are generated by scientific facilities worldwide. Often, these data need to be moved to other facilities in different geographic locations for analysis, archiving, distribution, and other purposes. For example, a single trillion-particle simulation with the Hardware/Hybrid Accelerated Cosmology Code (HACC) [1] generates 20 PB of raw data (500 snapshots, each 40 TB). These data are typically moved to other sites for analysis [2]. Research and education networks in different countries provide high-speed network connectivity between such facilities. For example, the U.S. Department of Energy's Energy Sciences Network (ESnet) provides connectivity to many science facilities at 100 Gbps or more. It carries around 20 petabytes monthly, primarily file transfers [3]. Science facilities employ dedicated data transfer nodes (DTNs) [4] for wide area file transfers.

GridFTP, an extension of the standard FTP protocol, is widely used for large science transfers, in particular via Globus [5], a cloud-hosted software-as-a-service that orchestrates file transfers between pairs of storage systems, mounted typically on dedicated data transfer nodes running GridFTP servers. (Other tools used for file transfers include FTP, rsync, SCP, BBCP [6], FDT [7], XDD, and Aspera.) GridFTP provides high performance, improved security relative to FTP, and improved reliability. Depending on the number and sizes of files in a user transfer request, Globus uses a heuristic to determine the number of GridFTP server processes (potentially on different DTNs, if a site has more than one DTN) to use. This number is often referred to as concurrency. Pipeline depth, another optimization parameter, specifies the number of files to be queued in each GridFTP server process in advance [2]. These concurrency and pipeline depth parameters influence the number of files (in a transfer request) assigned to each server process. Each GridFTP server process uses multiple TCP connections to transfer each file, caching these TCP connections and reusing them for all file transfer requests in a single client session.

In previous work [3], we characterized approximately 40 billion files totaling 3.3 exabytes transferred with GridFTP and 4.8 million file collections transferred by using the Globus transfer service, both during the period 2014-01-01 to 2018-01-01. Figure 1 shows the cumulative distribution of total bytes transferred by Globus during this period as a function of the number of files in a transfer. We see that 90% of all bytes transferred are from transfer requests with more than one file (the starting point of the number of files in a transfer is one in Figure 1), 63% are from transfers with more than 1,000 files, and 42% are from transfers with 10,000 or more files. Thus, it is important to get insights into the performance bottlenecks for multifile transfers: the focus of the work reported here.

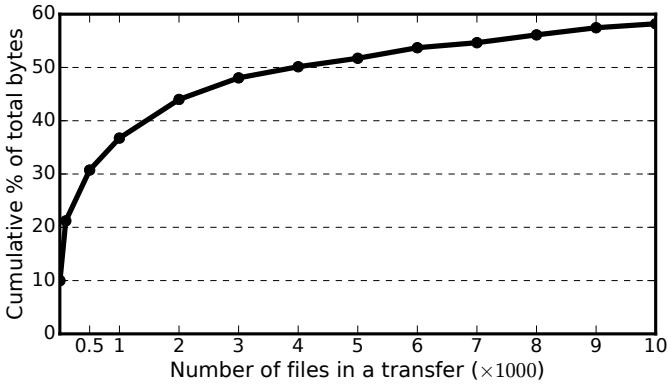


Fig. 1: Cumulative distribution of total bytes transferred using Globus by the number of files in a transfer, from the year 2014 to 2017.

The rest of the paper is as follows. §II provides the background and motivation for the work. In §III we analyze bottlenecks for multifile transfers between two high-performance computing (HPC) facilities. Then in §IV, we use regression to measure indirectly the per-file overhead of each major subsystem involved in these transfers. In §V we study the effectiveness of concurrent transfers in suppressing per-file overheads. As an alternative to using high concurrency to hide per-file overhead, §VI presents a prototype prefetching mechanism. In §VII we review related work, and in §VIII we summarize our conclusions.

II. Background

The ability to move large volumes of data rapidly between supercomputer facilities is critical for a (growing) number of science projects [2], [8]. The Petascale DTN collaboration, comprising staff at ESnet and four supercomputing facilities, namely, the National Energy Research Scientific Computing Center (NERSC), Argonne Leadership Computing Facility (ALCF), Oak Ridge Leadership Computing Facility (OLCF), and National Center for Supercomputing Applications (NCSA), was formed in 2016 to enhance data transfer rates among these facilities. Its goal was to achieve routine wide area file transfer rates of 15 Gbps, so as to enable the movement of one petabyte between any two of these major facilities in one week.

TABLE I: Data transfer rates (Gbps) among four major supercomputing facilities as various optimizations were applied over time. Data source: [9].

(a) March 2016

Destination \ *Source*	ALCF	NCSA	NERSC	OLCF
ALCF	–	8.2	7.3	11.1
NCSA	13.4	–	7.6	13.3
NERSC	10.0	6.8	–	6.7
OLCF	10.5	6.9	6.0	–

(b) June 2017

Destination \ *Source*	ALCF	NCSA	NERSC	OLCF
ALCF	–	21.2	27.2	23.0
NCSA	19.4	–	20.2	15.1
NERSC	22.9	11.8	–	19.7
OLCF	25.7	15.2	20.6	–

(c) November 2017

Destination \ *Source*	ALCF	NCSA	NERSC	OLCF
ALCF	–	56.7	42.2	47.5
NCSA	50.0	–	33.7	43.4
NERSC	35.0	22.6	–	33.1
OLCF	46.8	34.7	39.0	–

Petascale DTN project team members used the Globus transfer service to move a cosmology data set (referred to as L380) repeatedly between the four facilities. This dataset comprises about 4.4 terabytes of small- and medium-sized files; transfers use the automatic performance tuning heuristics in the Globus transfer service to set concurrency and pipeline depth parameters.

Table Ia shows file transfer rates observed when the project started in 2016. We see that none of the throughput meet the project goal of 15 Gbps throughput: a few links were within 10-15% of the goal, but others were significantly lower. To achieve the goal, collaboration members made a number of improvements to the data transfer infrastructure, such as updating DTN software, changing DTN configurations, and adding new DTNs. Table Ib shows the file transfer rates among various pairs of the four facilities 15 months after the Petascale DTN project started. All but one of the links among the four facilities achieved sustained file transfer rates above the target rate of one petabyte per week.

Since autumn 2017, Globus teams have worked on an optimization to the transfer service that improves load balancing and reduces the problem of long tails, that is, the situation where some of the concurrent GridFTP server processes involved in a transfer take much longer to process the files allocated to them than do others [10]. This optimization sorts the files to be transferred in descending order of their size before assigning them to the GridFTP server processes. In addition, for the last $10\times C$ files (where C is the concurrency: i.e., the number of GridFTP server processes), this optimization forces a pipeline depth of 1. The overall effect of these changes is to reduce the long tail effect, and as shown in Table Ic, to improve transfer rates by factors ranging from 1.5 to 3.5.

III. Bottleneck Analysis

The performance improvements reported in the preceding section are impressive, but we aspire to do even better. To that end, we seek to understand the bottleneck(s) in the end-to-end transfers. Here, we first describe the testbed, dataset, and process that we used to perform the bottleneck analysis, and we then present our analysis results and inferences.

A. Testbed

For the bottleneck analysis, we picked two of the four sites involved in the Petascale DTN project, namely, ALCF and NERSC, because we already had access to their DTNs. The ALCF has a 7 PB GPFS and NERSC a 28 PB Lustre filesystem. There are multiple 100 Gbps wide area connections between the ALCF and ESNet and multiple 100 Gbps connections between NERSC and ESNet, but the end-to-end bandwidth between the ALCF and NERSC is 100 Gbps. The round-trip time between the ALCF and NERSC is about 45 ms. The ALCF has 12 DTNs, each with one Intel Xeon E5-2667 v4 @3.20 GHz CPU, 64 GB of RAM, and one 10 Gbps NIC. NERSC has 10 DTNs, each of which has two Intel Xeon E5-2680 v2 @2.80 GHz CPU, 128 GB of RAM ,and 2 $\times$ 10 Gbps NIC.

B. Experiments

We performed a range of experiments to measure the performance of both end-to-end file transfers and the major subsystems involved in transfers, namely, the source storage

system, the destination storage system, and the network in between:

1) *Read-bench* measures storage read throughput by reading data from the parallel file system (PFS) to memory.
2) *Write-bench* measures storage write throughput by writing data from memory to PFS.
3) *Read-bench-G* measures storage read throughput by using GridFTP to transfer files from PFS to */dev/null* locally (source and destination are the same).
4) *Write-bench-G* measures storage write throughput by using GridFTP to perform local (i.e., same source and destination) transfers from */dev/zero* to PFS.
5) *Net-bench-G* performs memory-to-memory GridFTP transfers to measure wide area network (WAN) performance. This experiment involves multiple equal-sized transfers from */dev/zero* at the source to */dev/null* at the destination, performed concurrently with GridFTP.
6) *F2F* measures file-to-file GridFTP transfers over the WAN. All subsystems are involved in this experiment.
7) *F2M* measures file-to-memory GridFTP transfer over the WAN. Destination storage is not involved in this experiment.

We performed the GridFTP tests in this list with the `globus-url-copy` [11] command line client. This tool lacks many of the features of the Globus transfer service (hosted GridFTP client) and is less user-friendly but is more convenient for experimental purposes. `globus-url-copy` and the Globus transfer service are interchangeable in most of our experiments.

C. Dataset

A typical user transfer consists of one or more files and zero or more folders. Before we undertake the bottleneck analysis, we want to verify that the dataset used for the analysis is representative of science wide area file transfers, in order that our findings and proposed optimization methods can benefit other such transfers.

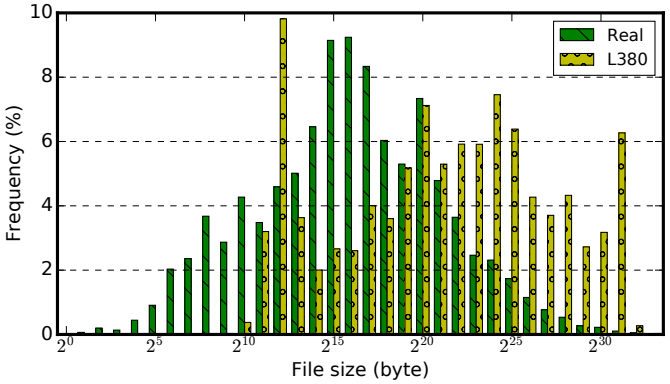


Fig. 2: Distribution of file sizes in the dataset used by the Petascale DTN project vs. a larger representative set of GridFTP transfers

Figure 2 compares the file size distributions of (a) the L380 dataset used in the Petascale DTN project (Table I) and (b) all transfers from 2014 to 2017 among more than 64,000 Globus GridFTP servers worldwide [3]. We see that the L380 file size distribution is not representative of typical GridFTP transfers. Thus, for the experiments reported here we generate a synthetic dataset with a file size distribution similar to that of all production GridFTP transfers. We refer to this dataset, which comprises 59,589 files totaling 1 TB, as DS_{real} throughout the paper. We note that given the hardware specification of our testbed, we need a large dataset so that the transfer time will be long enough for representativeness. For other scenarios, the dataset size can be varied by simply adjusting the number of files sampled in the distribution.

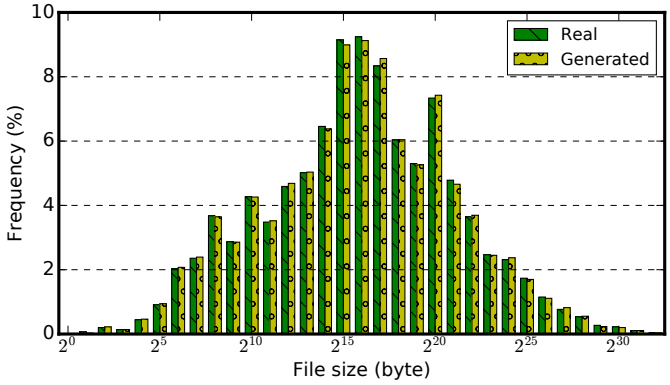


Fig. 3: Distribution of dataset file size, generated versus real.

Considering the representativeness of the file size distribution, we believe that this distribution, which was extracted from massive production GridFTP transfer logs, will be useful for benchmarking the performance of file transfer tools and for estimating the real benefits of file transfer optimizations. The distribution and code to generate files that follow the distribution are openly available at https://github.com/ramsesproject/transfer-file-dist.

D. Influence of dataset characteristics on performance

Figure 4 compares the performance of L380 and DS_{real}. One can see that the performance of DS_{real} is significantly lower than that of L380, which indicates that dataset characteristics can have a significant impact on the performance. In order to further establish the relationship between the dataset characteristics and the performance, we created a dataset of the same size as DS_{real} but having just enough files (i.e., the same as the concurrency used) to utilize all the concurrent processes used for data transfer using Globus. We refer to this dataset as DS_{big}. We kept the size of all files in DS_{big} the same to avoid any tail effects.

We transferred DS_{real} and DS_{big} one after the other using the Globus transfer service and repeated the process10 times to reduce the influence of randomness and gather statistically significant results. We used the default endpoint configuration of the ALCF and NERSC DTNs in Globus (i.e., 64 concurrent transfers and 4 TCP streams for each transfer). Figure 4 compares their performance. We see that the file size

characteristics and number of files have significant influence on transfer performance.

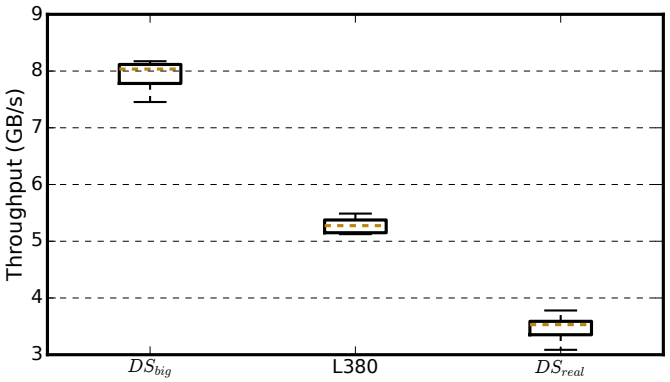


Fig. 4: Comparison of transfer performance for the DS_{big}, L380, and DS_{real} datasets between the ALCF and NERSC. Each transfer was repeated 10 times.

We note that the WAN is not dedicated for our experiments; that is, other programs consumed some of the wide area bandwidth between the ALCF and NERSC. Even though we repeated our experiments several times, we used the Simple Network Management Protocol (SNMP) data provided by ESNet to make sure that DS_{real} was not so unlucky to always have external load (and DS_{big} was not so lucky to never have any external load) during the tests. ESNet provides the throughput of its routers' interface [12] every 30 seconds. Figure 5 shows the network routers' interface on the `traceroute` (a program that tracks the network path of data between two points) between the ALCF and NERSC.

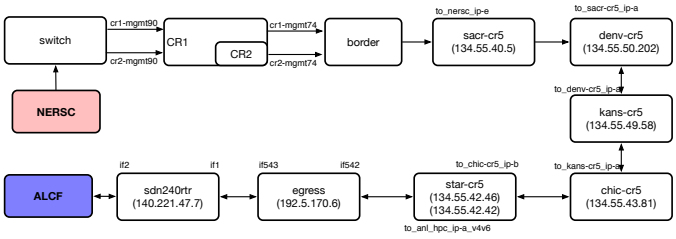


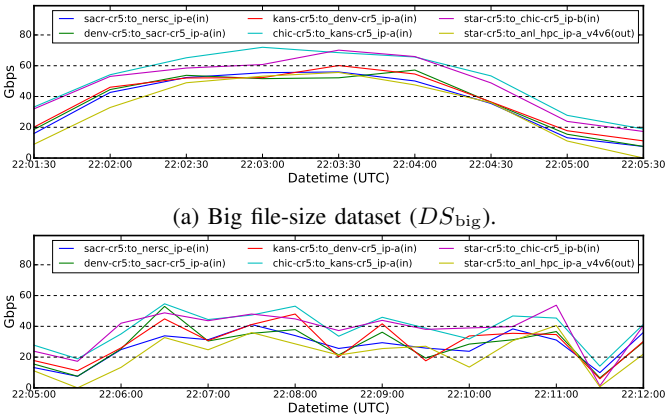
Fig. 5: Routing path between the ALCF and NERSC.

Figure 6 shows the throughput of all router interfaces between the ALCF and NERSC during our transfer of DS_{big} and DS_{real}, respectively.

All interfaces have a 100 Gbps capability. Clearly the network is not saturated during the transfer, thus indicating that the lower performance of DS_{real} is not due to external network load.

We conducted a subset of experiments described in §III-B to identify the bottleneck. Figure 7 shows the benchmark results for both DS_{big} and DS_{real}. The same block size is used for *Read-bench*, *Read-bench-G*, *Write-bench*, and *Write-bench-G*.

`iPerf3` is commonly used for benchmarking network performance, but it cannot emulate multifile transfers and in particular does not reuse TCP connections across files, as



(a) Big file-size dataset (DS_{big}).

(b) Real file-size dataset (DS_{real}).

Fig. 6: Router interface throughput when transfer DS_{big} and DS_{real} separately using Globus.

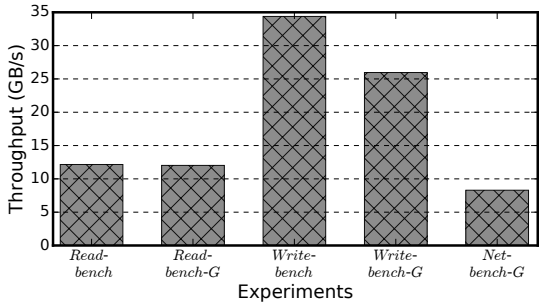
GridFTP does. So, first we compared the performance of `iPerf3` with that of */dev/zero* to */dev/null* GridFTP transfers from source to destination for a fixed duration. Both tests obtained similar performance. Figure 8 and Figure 9 show the throughput of all corresponding router interfaces in the WAN for both tests. Clearly, the transfer tool can have capability similar to that of `iPerf3` to saturate the network. Thus, we benchmarked the performance of the WAN between the NERSC and the ALCF using GridFTP by transferring the same amount of bytes and files as DS_{big} and DS_{real} from */dev/zero* at the source to */dev/null* at the destination (to avoid the impact of storage system overhead on network performance).

From Figure 7, we can see that the bottleneck is in fact the network and not the source or destination storage for both the DS_{big} and DS_{real} datasets. Even though the network is the ultimate bottleneck for DS_{real} dataset, Figure 7 shows a noticeable drop in performance for DS_{real} compared with DS_{big} for each case benchmarked. This establishes that there is a per-file overhead in storage read, storage write, and the network. In the following section, we use regression to provide an in-depth analysis of the per-file overhead introduced by each of these components.

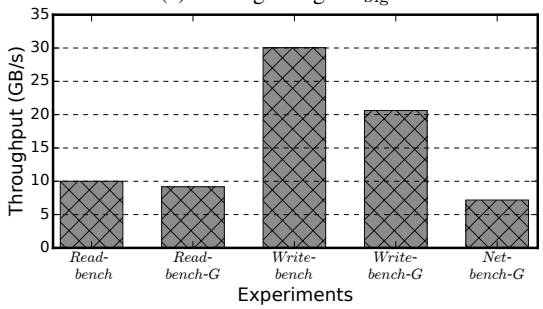
IV. Further Insights

Clearly, the per-file overheads cause the observed performance degradation. Direct measurement of these overhead requires instrumenting the transfer tool, which would make this work specific to the tool. Also, deploying a new (instrumented) version of a tool in the production environment used for our testing is challenging. Thus, we performed experiments using one concurrency to measure the overall per-file overhead indirectly. To identify directions for optimization, we also broke down the per-file overhead for each subsystem as follows:

- *Storage read overhead* is introduced by (previous) file close and (next) file open at the source (O_R);



(a) Testing using DS_{big}



(b) Testing using DS_{real}.

Fig. 7: Storage and network benchmark for file transfer. The network is benchmarked by transferring N equally sized */dev/zero* at NERSC to */dev/null* at the ALCF, where N is the number of files of DS_{big} and DS_{real}, respectively.

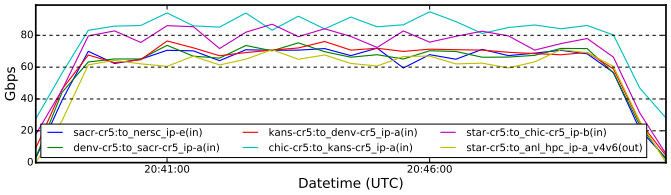


Fig. 8: Router interface throughput during the memory-to-memory transfer using Globus.

- *Storage write overhead* is introduced by (previous) file close and (next) file open at the destination (O_W);
- *Network overhead* is caused by TCP dynamics due to discontinuity in data flow caused by O_R and/or O_W (O_N);

If there were an infinite amount of buffer in each subsystem, the overall per-file overhead would be the maximum of O_R, O_N and O_W. With no buffers at all, it would be the sum of O_R, O_N and O_W. With limited buffers, the overall per-file overhead will be between max(O_R, O_N, O_W) and O_R + O_N + O_W.

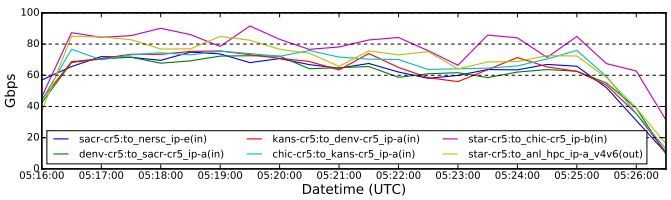


Fig. 9: Router interface throughput during the `iPerf3` test.

Assuming that each file introduces a fixed overhead of t_0, the network throughput is R. Thus, the time T to transfer N files totaling B bytes with one concurrency will be

$$T = N * t_0 + \frac{B}{R}. \tag{1}$$

Note that this model holds true only if each file introduces overhead that is independent of file size. To verify this model and measure the overhead of a file indirectly, we performed a series of experiments. We kept the total dataset size the same (5 GB, chosen to keep the experiment time short to reduce the influence of external load) for all experiments but varied the number of files in each experiment. We then transferred these datasets from NERSC to the ALCF using `globus-url-copy`. Figure 10 shows the results. A clear linear relation exists between the transfer time and the number of files. The Pearson correlation coefficient [13] between T and N is 0.995, indicating that they have a strong linear correlation. Thus, the experiment results verified that the model shown in Equation 1 holds true. We therefore can use regression analysis to fit parameters t_0 and $\frac{B}{R}$ as a combination. The coefficients are as follows:

$$T = 0.0665N + 16.5, \tag{2}$$

implying that the per-file overhead is 66.5 ms and that this overhead is the cause for the performance drop (seen in Figure 4) for the dataset with many files.

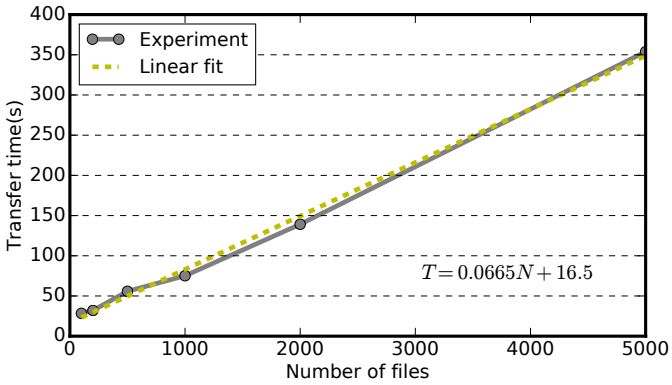
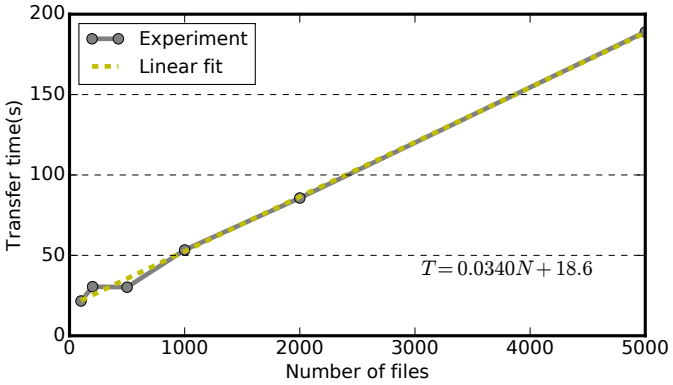


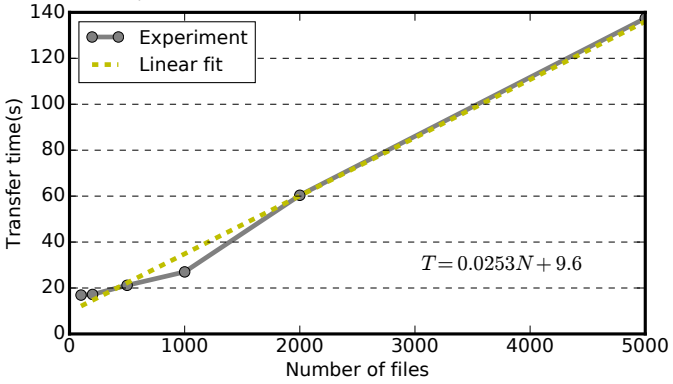
Fig. 10: Transfer time as a function of the number of files for transfer of files between NERSC and the ALCF. Transfer size is 5 GB.

To measure O_R, O_N, and O_W, we performed transfer experiments from files to */dev/null* at NERSC, from */dev/zero* on NERSC to */dev/null* on the ALCF and from */dev/zero* to files at the ALCF. All these experiments used a total dataset of 5 GB with varying numbers of files (the same as that of experiments corresponding to Figure 10. Thus, each regression analysis revealed the per-file overhead of that particular subsystem. Figure 11 shows the experiment results.

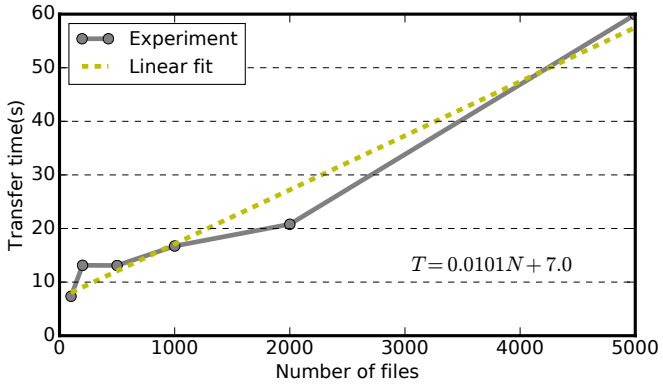
The regression models are T = 0.034N + 18.6, T = 0.0253N + 9.6, and T = 0.0101N + 7.0 for storage read, network, and storage write, respectively. O_R, O_N, and O_W are 34.0 ms, 25.3 ms, and 10.1 ms, respectively.



(a) Transfer time as a function of number of files for files to */dev/null* transfer locally at NERSC. Transfer size is 5 GB.



(b) Transfer time as a function of number of files for */dev/zero* to */dev/null* transfer over WAN between NERSC and the ALCF. Transfer size is 5 GB.



(c) Transfer time as a function of number of files for */dev/zero* to files transfer locally at the ALCF. Transfer size is 5 GB.

Fig. 11: Per-file overhead analysis using regression.

We note that O_N also includes the cost of opening and closing */dev/zero* and */dev/null*, operations that were performed many times because of the large number of files in the dataset. To evaluate the significance of this cost, we performed the */dev/zero* to */dev/null* transfers locally at NERSC. Figure 12 shows the results. We find that the per-file cost introduced by opening and closing of */dev/zero* and */dev/null* is only 0.3 ms. So, O_N is still 25 ms.

Thus, as we hypothesized in the beginning of this section,

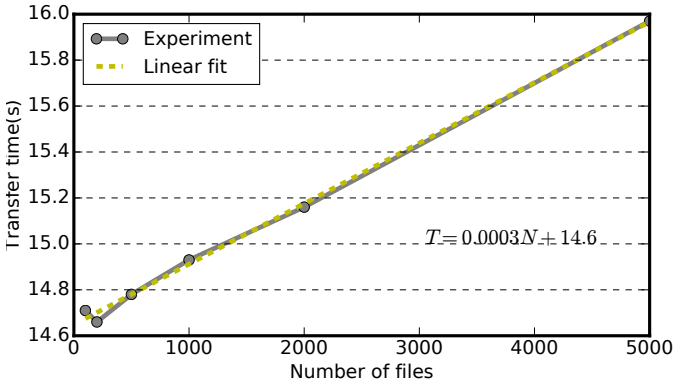


Fig. 12: Transfer time vs. number of files for 5 GB local transfer from */dev/zero* to */dev/null* at NERSC.

the per-file overhead for end-to-end file transfer (66.5 ms) lies between the maximum of the individual subsystems per-file overhead (34.0 ms) and the sum of per-file overhead of each individual subsystem (69.1 ms) and is much closer to the latter in practice.

The per-file overhead is significant and needs to be reduced in order to achieve high performance for transfers with large number of files. Prefetching the files can help reduce O_R and thus can indirectly reduce O_N, but it requires changes to the transfer tool. Concurrent transfer of multiple files can help reduce the average per-file overhead.

V. Concurrent Transfers

It is well known that concurrent transfers will help improve the performance of transfers with many files [14], [15], [2]. Usually, the performance improves with increasing concurrency only to a certain concurrency value. Increasing concurrency beyond that value will hurt the transfer performance [14], [15].

In §IV, we saw that each subsystem operation (storage read, network transfer and storage write) has a fixed overhead for each file transferred. Among them, storage read has the highest overhead in our experimental environment. The subsystem that has the highest overhead will vary based on the environment but we expect each subsystem to have a per-file overhead. In this section, we study how concurrent transfers of multiple files can help reduce the average per-file overhead for each subsystem. Since increasing concurrency will result in negative returns beyond a certain value (because of contention) and this value varies based on external factors (such as external load on network and storage), determining the "just right" concurrency is hard [15], [16]. Still, we perform transfer experiments with the representative dataset DS_{real} from NERSC to ALCF to understand the impact of concurrent transfers in reducing the average per-file overhead.

A. Storage read

We used `globus-url-copy` [11] to transfer DS_{real} from the parallel file system at NERSC to */dev/null* locally with varying numbers of concurrent file transfers. Figure 13 shows

the throughput as a function of concurrency. Throughput increases with the number of concurrency up to 14 GB/s; then, further increasing concurrency does not help.

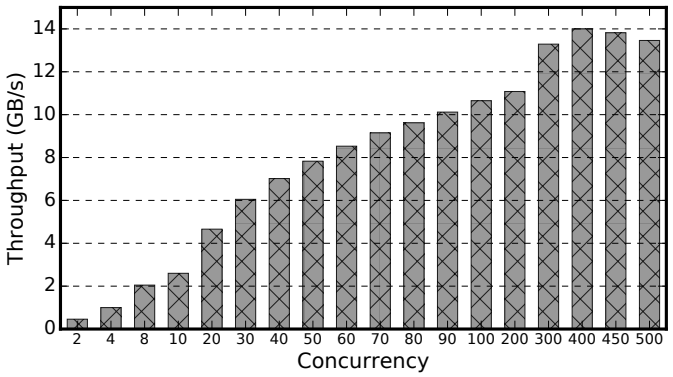


Fig. 13: *Read-bench-G*: Lustre read performance test using `globus-url-copy`.

B. Network

We next varied the concurrency to study the effectiveness of concurrency in suppressing per-file network overheads. We see from Figure 14 that per-file overheads can be suppressed with sufficient concurrency. The amount of concurrency needed to achieve the maximum throughput is much smaller than that needed to suppress the per-file read overhead (as shown in Figure 13).

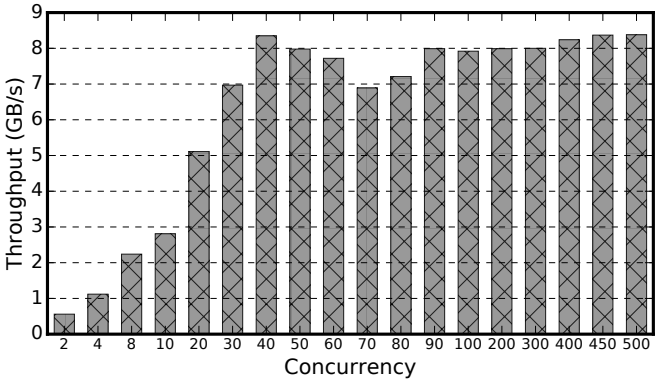


Fig. 14: *Net-bench-G*: Transfer 59,589 */dev/zero* totaling 1 TB (the same as DS_{real}) at NERSC DTNs to */dev/null* at the ALCF DTNs.

Experiments in §V-A (i.e., Figure 13) involve only file read operations. Here we add the network to experiments in §V-A by transferring DS_{real} from NERSC to */dev/null* at the ALCF. Figure 15 shows throughput as a function of concurrency. Here again throughput increases with concurrency up to point and then starts decreasing.

C. Storage write

We next transfer data from */dev/zero* to the parallel file system locally at the ALCF. Specifically, we write 59,589 equal-sized files, totaling 1 TB, because of a limitation in the transfer tool that did not allow us to transfer varying lengths of data from */dev/zero* in a single transfer. As discussed in §IV, the per-file overhead is fixed and independent of file size. Thus, the per-file overhead for this dataset will be the same as for DS_{real}. Figure 16 shows the results of this experiment. Here again we see diminishing returns beyond a certain point, albeit with a strange pattern.

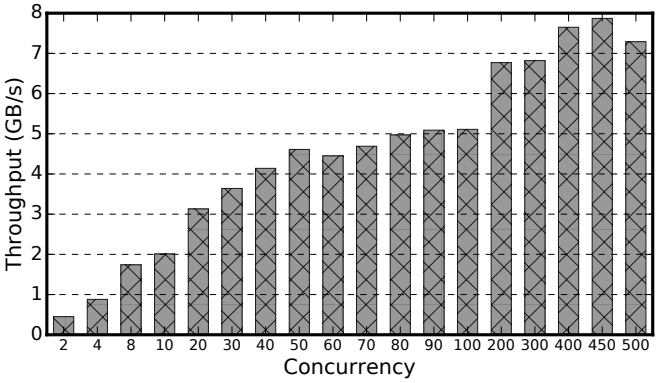


Fig. 15: *F2M*: Transfer files on Lustre at NERSC to */dev/null* at the ALCF DTNs.

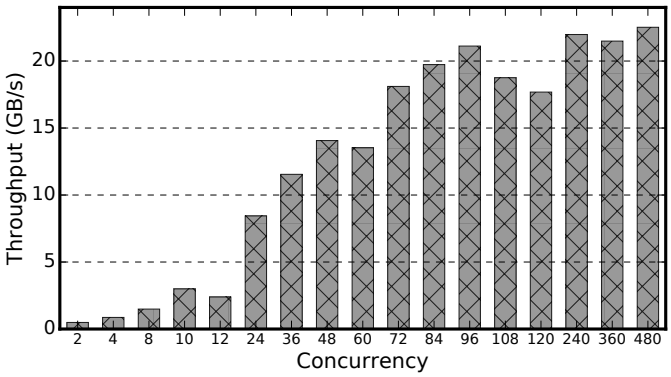


Fig. 16: *Write-bench-G*: Transfer from */dev/zero* at the ALCF DTNs to files on GPFS at the ALCF for the GPFS write performance benchmark.

D. End-to-end file transfers

Here we present the results of transferring DS_{real} from the parallel file system at NERSC to the parallel file system at the ALCF using `globus-url-copy` with different concurrency values. Figure 17 shows the performance results.

We can see that the throughputs in Figure 17 are slightly worse than that of the corresponding throughputs in Figure 15 for the most part (for extremely high concurrency values, it is not true, but we attribute that to noise). The network is the bottleneck in both cases, but Figure 17 can have additional per-file overhead up to O_W, which is 10.1 ms.

Overall, these experiments show that datasets with many files can achieve the maximum performance (as achieved by a dataset with an optimal number of files) by using a "just right" concurrency (i.e., about 450 in this case). In other words, by

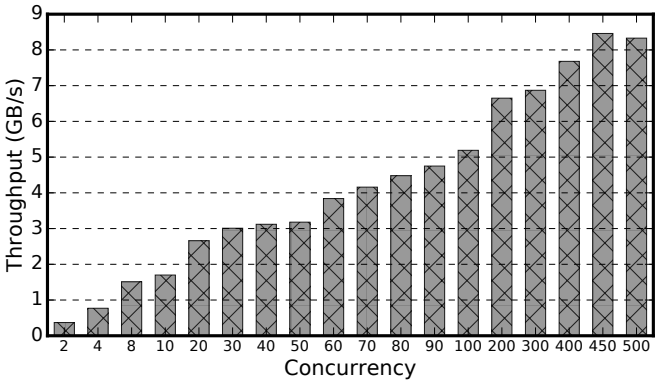


Fig. 17: *F2F*: Transfer files on Lustre at NERSC to GPFS at the ALCF.

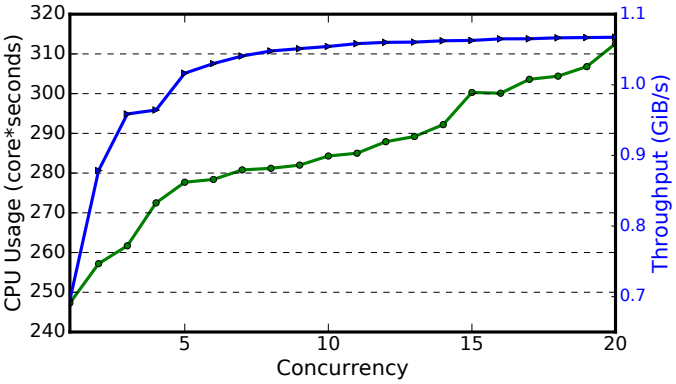


Fig. 19: CPU utilization vs. transfer concurrency.

adjusting the number of concurrent transfers, the maximum performance for wide area data transfers can be achieved irrespective of the dataset characteristics. Identifying the "just right" concurrency is a challenging task especially because it varies based on the environmental conditions and also different subsystems have different optimal concurrency values. For example, as can be observed from Figure 18, even between the same pair of endpoints the "just right" concurrency values vary based on the direction of the transfers.

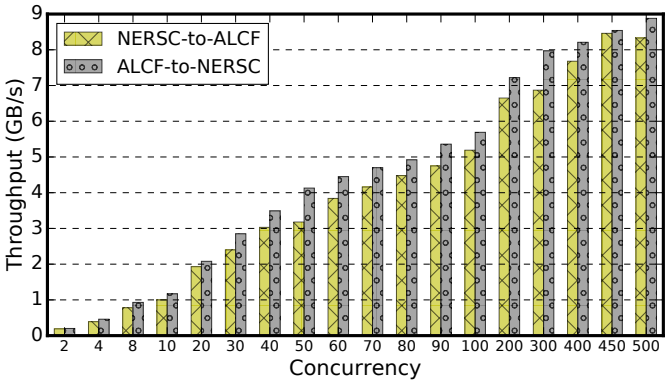


Fig. 18: Comparison of performance when transfer DS_{real} from the ALCF to NERSC and the reverse direction.

VI. Prefetching

From the preceding section, it is clear that the per-file overhead can be reduced by concurrently transferring multiple files. We also note that the per-file overhead can even be made negligible (if not completely eliminated), but the amount of concurrency needed to achieve that is too high (> 400). For example, Figure 19 shows the total CPU utilization (in core*seconds) to transfer a given dataset with different concurrency. Although high levels of concurrency achieve better performance, they consume more CPU as well and thus can negatively impact other transfers.

Thus, another approach to reduce the per-file overhead is prefetching. In this section, we explore a prefetching mechanism to alleviate the impact of O_R (and thus O_N at least to some extent). We recall that O_R is ~ 34 ms and O_N is ~ 25 ms. GridFTP utilizes pipelining to avoid communication overhead between the GridFTP server and client after completion of each file transfer. However, overhead still exists between two files, which is the sum of (1) time taken to close the file whose transfer just got completed; (2) time to move the disk probe to the location of the file that needs to be transferred next (`Nextfile`); and (3) time to open `Nextfile`.

Figure 20 shows the procedure of our prefetching-enabled algorithm. The idea is to prefetch one or more blocks of the `Nextfile` during the transfer of a file so that we can start transferring the `Nextfile` immediately upon completion of the ongoing file transfer, avoiding the overhead mentioned above. To ensure that the prefetching of `Nextfile` does not influence the ongoing file transfer, we do the prefetching only when the ongoing transfer has filled the TCP send buffer (and is waiting for space in the buffer to write the next block). Every time the ongoing transfer fills the TCP buffer, we read a block of data (256 KB) of `Nextfile` (or the following file(s) if we still have space in the prefetch buffer

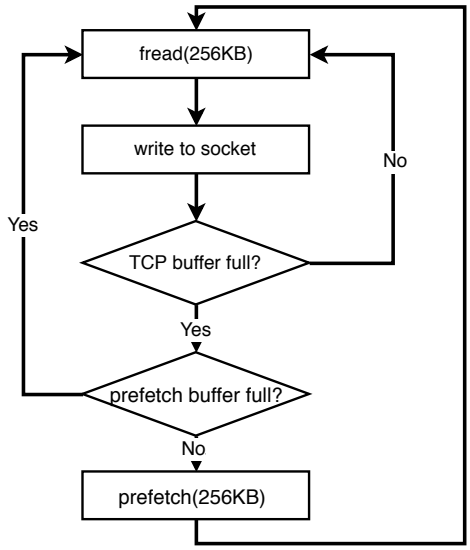


Fig. 20: Flow diagram of the prefetching approach

after fetching `Nextfile` in its entirety). This prefetch buffer size is configurable: we used 2 MB.

Figure 21 shows the effectiveness of prefetching using multiple 80 GB transfers, each with a different number of files. We note that the transfer time increases much more slowly with increasing number of files when prefetching is enabled. Thus, prefetching can help reduce the per-file overhead significantly.

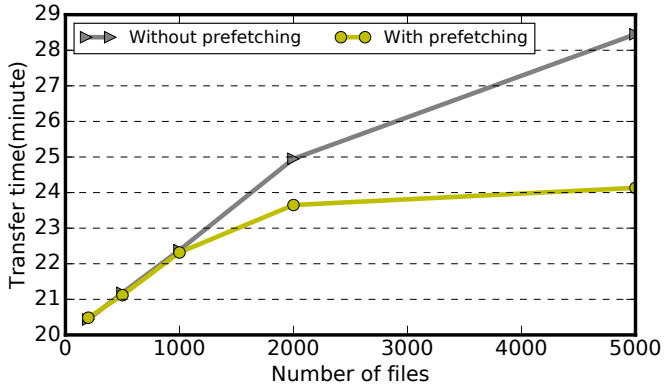


Fig. 21: Transfer time as a function of number of files for 80 GB transfers from NERSC to the ALCF.

Figure 22 shows the throughput for 2 TB dataset (containing 50,000 files) transfers with and without prefetching for different concurrency values. Prefetching clearly helps achieve a

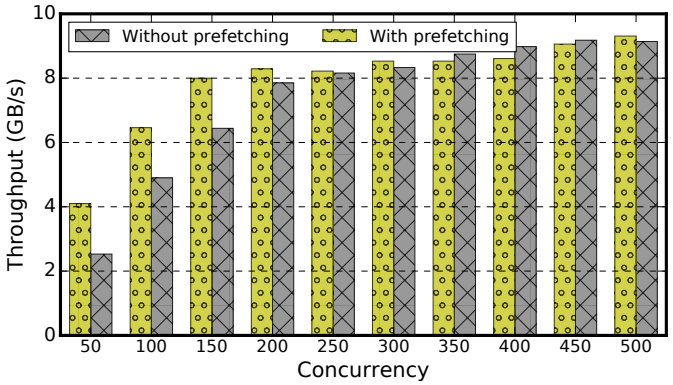


Fig. 22: Transfer files on Lustre at NERSC to GPFS at the ALCF.

higher throughput with less concurrency.

We note that experiments in Figure 22 do not use parallel TCP streams for transferring each file, whereas the experiments in Figure 17 uses four parallel streams for each file transfer. This is the primary reason we still need a reasonably high concurrency (although it is much lower when compared with no prefetching) to achieve a high throughput.

VII. Related Work

Many publications examine methods to optimize the performance of large data transfers over wide area networks by tuning application-level parameters. Liu et al. [17] developed a tool to optimize multifile transfers by opening multiple GridFTP threads. Their tool increases the number of concurrent flows up to the point where the transfer performance degrades. Their work also proved that the number of concurrent transfer files can benefit a transfer only to some extent, causing negative influence after that optimal point.

Kim et al. [18] proposed an application-layer throughput optimization model based on prediction of the appropriate number of parallel TCP streams. It relies on real-time network probing, which either causes too much sampling overhead or fails to predict accurately the correct transfer parameters for long-running transfers when network conditions vary over time.

Engin et al. [19] clustered files by size and then used a heuristic approach to estimate the optimal Globus application-level parameter values (i.e., pipelining, parallelism, and concurrency) to be used in each cluster, in order to maximize the overall transfer throughput in wide area networks. Specifically, based on file characteristics and real-time investigation, their algorithms dynamically tune parallelism per file, the level of control channel pipelining, and the number of concurrent file transfers to increase I/O throughput. In another work, Engin et al. [20] combined historical data analysis with real-time sampling to enable their algorithms to tune the application-level data transfer parameters accurately and efficiently, in order to achieve close-to-optimal end-to-end data transfer throughput with low overhead.

Nine et al. [21] used historical knowledge about the network and data to reduce the real-time investigation overhead while ensuring near-optimal throughput for each transfer. However, the dataset used in the work to benchmark the tool is not sufficiently representative. As we presented in prior work [14], dataset characteristics have a significant influence on transfer performance. Furthermore, as we presented in this paper, even the same dataset may need different concurrency choices for different direction of two endpoints.

Rao et al. [22] studied the performance of TCP variants and their parameters for high-performance transfers over dedicated connections by collecting systematic measurements using physical and emulated dedicated connections. These experiments revealed important properties such as concave regions and relationships between dynamics and throughput profiles. Their analyses enable the selection of a high-throughput transport method and corresponding parameters for a given connection based on round-trip time.

VIII. Conclusion

It is well known that file transfer tools tend to perform better for datasets with a small number of large files than for datasets with many small files. This performance degradation has long been attributed to per-file overheads, but there has been little understanding on where exactly this overhead comes from and how different components in end-to-end file transfers contribute to this overhead. We have reported here on a thorough analysis of the per-file overheads introduced by the major components in wide area file transfers. We showed that

in an end-to-end file transfer, the per-file overhead is significantly higher than the maximum of the individual subsystems' per-file overhead. We also showed that the average per-file overhead can be reduced significantly for a representative dataset with many thousands of files by transferring many files concurrently, to the extent of achieving performance equivalent to that of a dataset with an optimal number of files. As an alternative to using high concurrency to reduce the per-file overhead, we showed that prefetching can reduce per-file overhead with much less concurrency.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357. Z. Liu and Y. Liu contributed equally to this research. We gratefully acknowledge the National Energy Research Scientific Computing Center and Argonne Leadership Computing Facility for providing us resources.

REFERENCES

[1] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, V. Vishwanath, Z. Lukić, S. Sehrish, and W. Liao, "HACC: Simulating sky surveys on state-of-the-art supercomputing architectures," *New Astronomy*, vol. 42, pp. 49–65, 2016.

[2] 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

[3] Z. Liu, R. Kettimuthu, I. Foster, and N. S. V. Rao, "Cross-geography scientific data transferring trends and behavior," in *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

[4] Nersc, *Science DMZ: Data Transfer Nodes*, 2018 (accessed December 12, 2018), http://fasterdata.es.net/science-dmz/DTN/.

[5] 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.

[6] "BBCP," http://www.slac.stanford.edu/~abh/bbcp/.

[7] FDT, *FDT - Fast Data Transfer*, 2018 (accessed January 3, 2018), http://monalisa.cern.ch/FDT/.

[8] 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

[9] Lawrence Berkeley National Laboratory, *ESnet's Petascale DTN Project Speeds up Data Transfers between Leading HPC Centers*, 2018 (accessed September 3, 2018), https://cs.lbl.gov/news-media/news/2017/esnets-petascale-dtn-project-speeds-up-data-transfers-between-leading-hpc-centers/.

[10] 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.

[11] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The Globus striped GridFTP framework and server," in *ACM/IEEE Conference on Supercomputing*, ser. SC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 54–. [Online]. Available: https://doi.org/10.1109/SC.2005.72

[12] *ESnet SNMP data*, 2018 (accessed September 3, 2018), https://graphite.es.net/west/.

[13] K. Pearson, "Note on regression and inheritance in the case of two parents," *Proceedings of the Royal Society of London*, vol. 58, pp. 240–242, 1895.

[14] Z. Liu, P. Balaprakash, R. Kettimuthu, and I. Foster, "Explaining wide area data transfer performance," in *26th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '17. New York, NY, USA: ACM, 2017, pp. 167–178. [Online]. Available: http://doi.acm.org/10.1145/3078597.3078605

[15] Z. Liu, R. Kettimuthu, I. Foster, and P. H. Beckman, "Toward a smart data transfer node," vol. 89, 2018, pp. 10–18. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18302346

[16] Z. Liu, R. Kettimuthu, P. Balaprakash, and I. Foster, "Building a wide-area data transfer performance predictor: An empirical study," in *the 1st International Conference on Machine Learning for Networking*, ser. MLN 2018. Springer, 2018.

[17] W. Liu, B. Tieman, R. Kettimuthu, and I. Foster, "A data transfer framework for large-scale science experiments," in *19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 717–724. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851582

[18] 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.

[19] E. Arslan and T. Kosar, "A heuristic approach to protocol tuning for high performance data transfers," *ArXiv e-prints*, Aug. 2017.

[20] E. Arslan, K. Guner, and T. Kosar, "HARP: predictive transfer optimization based on historical analysis and real-time probing," in *SC'16*, 2016, pp. 25:1–25:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=3014904.3014938

[21] M. S. Zulkar Nine, K. Guner, Z. Huang, X. Wang, J. Xu, and T. Kosar, "Data transfer optimization based on offline knowledge discovery and adaptive real-time sampling," *ArXiv e-prints*, Jul. 2017.

[22] N. S. Rao, Q. Liu, S. Sen, D. Towlsey, G. Vardoyan, R. Kettimuthu, and I. Foster, "TCP throughput profiles using measurements over dedicated connections," in *26th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2017, pp. 193–204.