Preprint published at the *Future Generation Computer Systems*. DOI: 10.1016/j.future.2018.05.051

Transferring a Petabyte in a Day[1]

Rajkumar Kettimuthu[a], Zhengchun Liu[a,*], David Wheeler[d], Ian Foster[a,b], Katrin Heitmann[a,c], Franck Cappello[a]

[a] *Mathematics & Computer Science Division, Argonne National Laboratory, Lemont, IL 60439, USA*
[b] *Department of Computer Science, University of Chicago, Chicago, IL 60637, USA*
[c] *High Energy Physics Division, Argonne National Laboratory, Lemont, IL 60439, USA*
[d] *National Center for Supercomputing Applications, University Of Illinois at Urbana–Champaign, Urbana, IL 61801, USA*

Abstract

Extreme-scale simulations and experiments can generate large amounts of data, whose volume can exceed the compute and/or storage capacity at the simulation or experimental facility. With the emergence of ultra-high-speed networks, researchers are considering pipelined approaches in which data are passed to a remote facility for analysis. Here we examine an extreme-scale cosmology simulation that, when run on a large fraction of a leadership computer, generates data at a rate of one petabyte per elapsed day. Writing those data to disk is inefficient and impractical, and in situ analysis poses its own difficulties. Thus we implement a pipeline in which data are generated on one supercomputer and then transferred, as they are generated, to a remote supercomputer for analysis. We use the Swift scripting language to instantiate this pipeline across Argonne National Laboratory and the National Center for Supercomputing Applications, which are connected by a 100 Gb/s network; and we demonstrate that by using the Globus transfer service we can achieve a sustained rate of 93 Gb/s over a 24-hour period, thus attaining our performance goal of one petabyte moved in 24 hours. This paper describes the methods used and summarizes the lessons learned in this demonstration.

Keywords: Wide area data transfer, GridFTP, Large data transfer, Cosmology workflow, Pipeline

1. Introduction

Extreme-scale scientific simulations and experiments can generate much more data than can be stored and analyzed efficiently at a single site. For example, a single trillion-particle simulation with the Hardware/Hybrid Accelerated Cosmology Code (HACC) [1] generates 20 PiB of raw data (500 snapshots, each 40 TiB), which is more than petascale systems such as the Mira system at the Argonne Leadership Computing Facility (ALCF) and the Blue Waters system at the National Center for Supercomputing Applications (NCSA) can store in their file systems. Moreover, as scientific instruments are optimized for specific objectives, both the computational infrastructure and the codes become more specialized as we reach the end of Moore's law. For example, one version of the HACC is optimized for the Mira supercomputer, on which it can scale to millions of cores, while the Blue Waters supercomputer is an excellent system for data analysis, because of its large memory (1.5 PiB) and 4000+ GPU accelerators. To demonstrate how we overcame the storage limitations and enabled the coordinated use of these two specialized systems, we conducted a pipelined remote analysis of HACC simulation data, as shown in Figure 1.

[1] a Petabyte (PiB) = 2^{15} bytes

* Corresponding author: Zhengchun Liu, Bldg. 240, Argonne National Lab., 9700 S. Cass Avenue, Lemont, IL 60439, USA
Email addresses: kettimut@anl.gov (Rajkumar Kettimuthu), zhengchun.liu@anl.gov (Zhengchun Liu), dwheeler@illinois.edu (David Wheeler), foster@anl.gov (Ian Foster), heitmann@anl.gov (Katrin Heitmann), cappello@mcs.anl.gov (Franck Cappello)

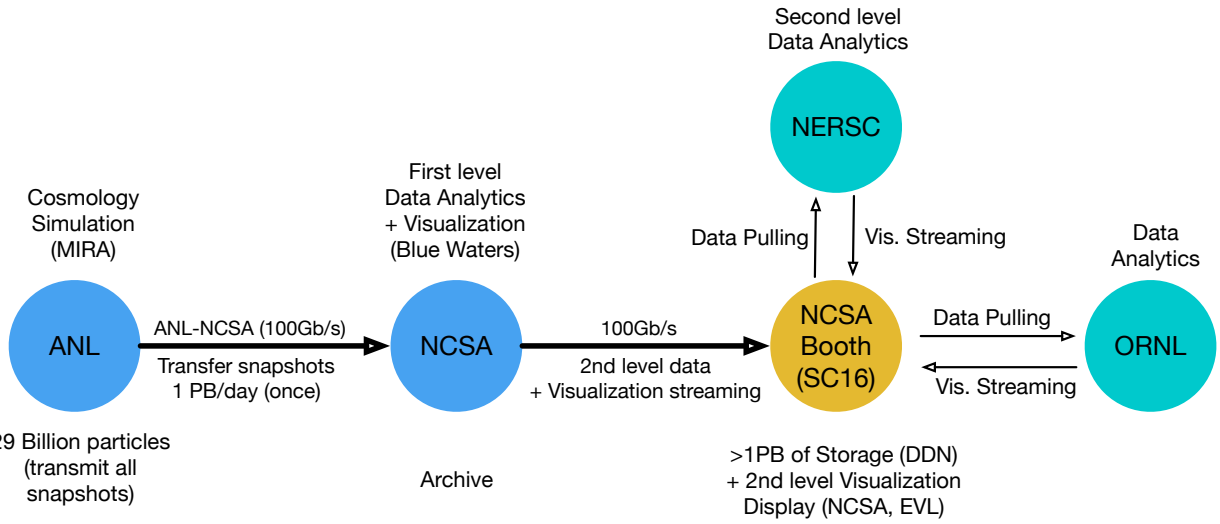

Figure 1: Pipelined execution of a cosmology workflow that involves a mix of streaming scenarios, ranging from sustained ∼100 Gb/s over a 24-hour period to high-bandwidth bursts during interactive analysis sessions.

Our demonstration at the NCSA booth of SC16 (the International Conference for High Performance Computing, Networking, Storage, and Analysis in 2016), a state-of-the-art, 29-billion-particle cosmology simulation combining high spatial and temporal resolution in a large cosmological volume was performed on Mira at ALCF. As this simulation ran, the Globus [2] transfer service was used to transmit simulation data to NCSA each of 500 temporal snapshots as it was produced [3]. In total, this workflow moved 1 PiB in 24 hours from the ALCF to NCSA, requiring an average end-to-end rate of ∼93 Gb/s. In this paper, we describe how we achieved this feat, including the experiments performed to gather insights on tuning parameters, data organization and the lessons we learned from the demonstration.

As snapshots arrived at NCSA, a first level of data analysis and visualization was performed using the GPU partition of Blue Waters. We note that the analysis tasks have to be carried out sequentially: information from the previous time snapshot is captured for the analysis of the next time snapshot in order to enable detailed tracking of the evolution of structures. The workflow system therefore was carefully designed to resubmit any unsuccessful analysis job and to wait for an analysis job to finish before starting the next one. The output data (half the size of the input data) was then sent to the NCSA booth at SC16 to allow access to and sharing of the resulting data from remote sites. The whole experiment was orchestrated by the Swift parallel scripting language [4]. In previous simulations, scientists were able to analyze only ∼100 snapshots because of infrastructure limitations.

This experiment achieved two objectives never accomplished before: (1) running a state-of-the-art cosmology simulation and analyzing all snapshots (currently only one in every five or 10 snapshots is stored or communicated); and (2) combining two different types of systems (simulation on Mira and data analytics on Blue Waters) that are geographically distributed and belong to different administrative domains to run an extreme-scale simulation and analyze the output in a pipelined fashion.

The work presented here is also unique in two other respects. First, while many previous studies have varied transfer parameters such as concurrency and parallelism in order to improve data transfer performance [5, 6, 7], we also demonstrate the value of varying the file size used for data transfer, which provides additional flexibility for optimization. Second, we demonstrate these methods in the context of dedicated data transfer nodes and a 100 Gb/s network circuit.

The rest of the paper is organized as follows. In §2 we introduce the science case and the environment in which we performed these transfers. In §3 we describe the tests to find the optimal transfer parameters. In §4 we summarize the performance of the transfers during the pipelined simulation and analysis experiments, and we describe our experiences with checksum-enabled transfers. Based on the demo at SC16, we propose

in §5 an analytical model to identify the optimal file size and show that it can help improve the performance of checksum-enabled transfers significantly. In §6 we review related work, and in §7 we summarize our work.

2. Science case and demonstration environment

We first present the challenges raised by the science problem and then describe the environment used for the demonstration.

2.1. Science case

To understand the Universe, cosmologists use large telescopes to conduct observational surveys. These surveys are becoming increasingly complex as telescopes reach deeper into space, mapping out the distributions of galaxies at farther distances. Cosmological simulations that track the detailed evolution of structure in the Universe over time are essential for interpreting these surveys. Cosmologists vary the fundamental physics in the simulations, evaluate resulting measurements in a controlled way, and then predict new phenomena. They also model systematic errors in the data, mimic inaccuracies due to limiting telescope and sensor artifacts, and determine how these limitations can influence scientific results. In order to achieve high-quality simulations, high temporal and spatial resolution are critical. Cosmologists need to track the evolution of small overdensities in detail and follow how they evolve into larger structures. Events early in the life of such a structure will determine what kind of galaxy it will host later, controlling, for example, the brightness, color, and morphology of the galaxy. Current (and next-generation) supercomputers (will) allow them to attain high spatial resolution in large cosmological volumes by simulating trillions of tracer particles. But the supercomputer on which the simulation is carried out might not be– and usually is not– the optimal system for data analysis because of storage limitation or supercomputer specialization.

2.2. Demonstration environment

For the demonstration, we ran a large simulation at the ALCF, moved the raw simulation output to NCSA, and ran an analysis program on the Blue Waters supercomputer. The simulation evolved 3072^3 (=29 billion) particles in a simulation box of volume $(512\mathrm{h}^{-1}\mathrm{Mpc})^3$. This led to an approximate mass resolution (mass of individual particles) of mp=$3.8*10^8$ h^{-1}Msun.

Each snapshot holds 1.2 TiB. The source of the data was the GPFS parallel file system on the Mira supercomputer at Argonne, and the destination was the Lustre parallel file system on the Blue Waters supercomputer at NCSA. Argonne and NCSA have 12 and 28 data transfer nodes (DTNs) [8], respectively, dedicated for wide area data transfer. Each DTN runs a Globus GridFTP server. We chose to use Globus to orchestrate our data transfers in order to get automatic fault recovery and load balancing among the available GridFTP servers on both ends.

It is clear that wide-area data transfer is central to distributed science. The data transfer time has direct influence on workflow performance and the transfer throughput estimation is crucial for workflow scheduling and resource allocation [9].

3. Exploration of tunable parameters

Concurrency, parallelism, and pipelining are three key performance optimization parameters for Globus GridFTP transfers. These effectiveness of the parameters depends on data transfer node, local storage system and network. There is not a one-size-fits-all setting that is optimal in any case [10]. Concurrency and parallelism mechanism are illustrated in Figure 2, and pipelining is illustrated in Figure 3.

3.1. Concurrency

Concurrency uses multiple GridFTP server processes at the source and destination, where each process transfers a separate file, and thus provides for concurrency at the file system I/O, CPU core, and network levels.

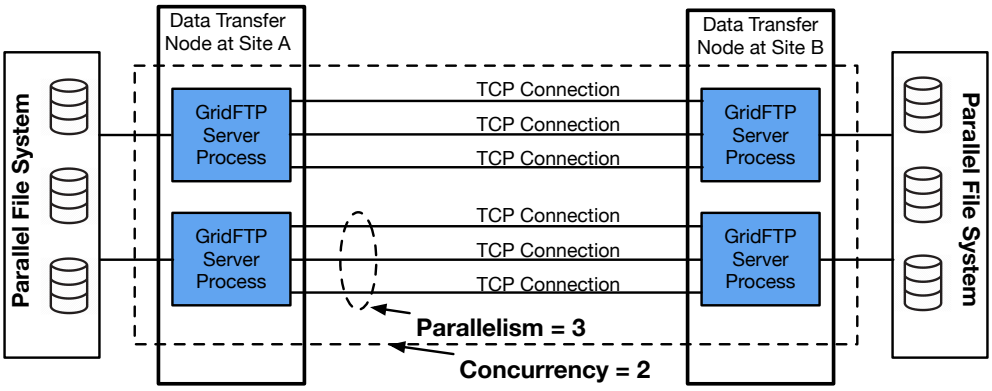

Figure 2: Illustration of concurrency and parallelism in Globus GridFTP.

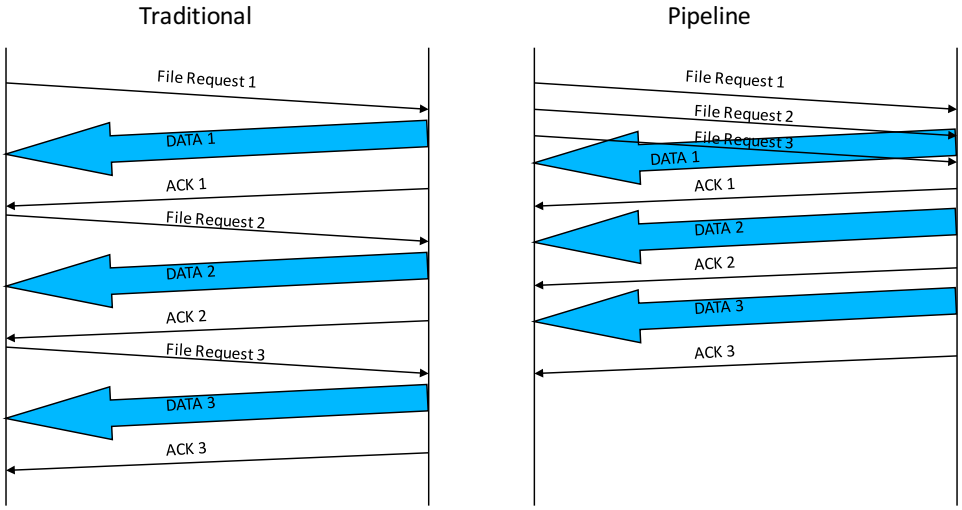

Figure 3: Illustration of pipelining in Globus GridFTP.

3.2. Parallelism

Parallelism is a network-level optimization that uses 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.

3.3. Pipelining

Pipelining speeds lots of tiny files by sending multiple FTP commands to a single GridFTP server process without waiting for the first command's response. This approach reduces latency between file transfers in a single GridFTP server process.

Thus, in order to find the best application-tunable parameters, we first arbitrarily fixed the average file size to be ∼4 GiB and evaluated different combinations of three Globus GridFTP parameters: concurrency, parallelism, and pipeline depth.

Figure 4 shows the achieved throughput as a function of parallelism for different concurrencies and pipeline depths. Parallelism clearly does not provide any obvious improvement in performance. We conjecture that the reason is that the round-trip time between source DTNs and destination DTNs is small (around 6 ms). Figure 5 shows the achieved throughput as a function of concurrency for different pipeline depths. We omitted parallelism in this comparison because it does not have much impact (Figure 4).

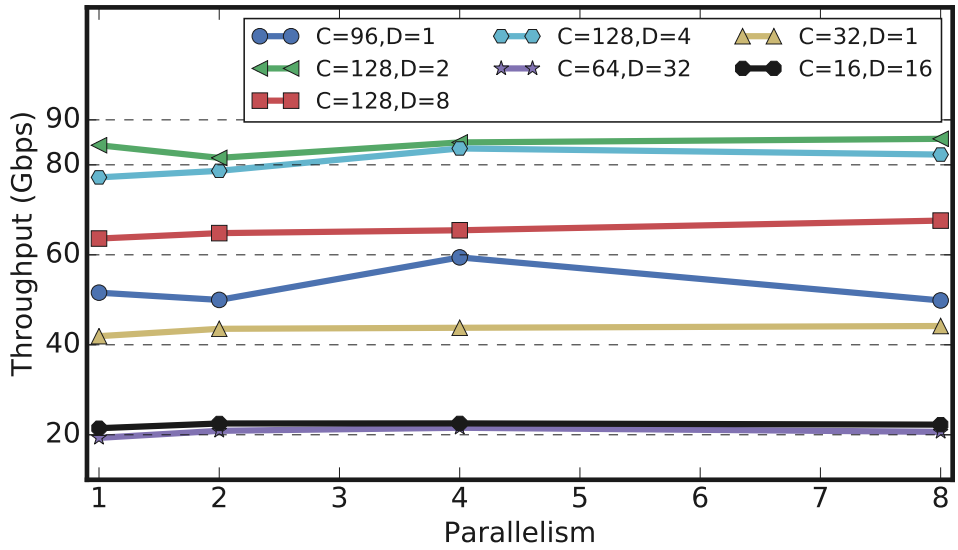

Figure 4: Data transfer performance versus parallelism with different concurrency (C) and pipeline depth(D).

From Figure 5, we see that the throughput increases with increasing concurrency, especially for small pipeline depths. Each DTN at ALCF has only a 10 Gb/s network interface card, and a transfer needs to use at least 10 DTNs to achieve the desired rate of 1 PiB in a day or sustained ∼93 Gb/s. In order for a transfer to use 10 DTNs, the concurrency has to be at least 10. In order for each DTN to drive close to 10 Gb/s (read data from the storage at 10 Gb/s and send data on the network at the same rate), many (or all) DTN cores need to be used. In this case, each DTN has 8 cores, and thus a concurrency of at least 96 is needed to use all cores. This explains why a concurrency of 128 gives the highest performance.

We also see in Figure 5 that increasing the pipeline depth reduces performance. The reason is that the Globus policy was designed for regular data transfers (i.e., transfer size is not as big as in this case, and the endpoints and network are not as powerful). Specifically, at the time of experiment, Globus doubles pipeline depth, and splits multi-file transfers into batches of 1,000 files and treats each batch like an independent transfer request. This policy was put in place to control the total number of concurrent file transfers and thus the memory load on the servers. For example, if we use a pipeline depth of 8, the maximum concurrency can only be $\lfloor \frac{1000}{16} \rfloor = 62$, which is why concurrency with 64 and 128 achieved the same performance in Figure 5. After feeding back findings in this paper, Globus has optimized this 1,000 files batch mechanism.

Similarly, when the pipeline depth is 16, the actual concurrency will be $\lfloor \frac{1000}{32} \rfloor = 31$, and thus transfers with concurrency greater than 32 achieve the same performance as those with 32 in Figure 5. Therefore, the optimal pipeline depth for our use case is 1, because pipelining is good for transferring tiny files (when the elapsed transfer time for one file is less than the round-trip time between the client and the source endpoint) but not for larger files.

Although we used Globus transfer service in this work, we believe that our methods and conclusions are applicable to other wide area data transfers. Because the three tunable parameters: concurrency (i.e., equivalent to the number of network and disk I/O threads / processes), parallelism (equivalent to the number of TCP connections) and pipeline (i.e., the number of control channel) of Globus are generally used by other high performance data transfer tools such as BBCP [11], FDT [12], dCache [13], FTS [14] and XDD [15].

4. Experiences transferring the data

As mentioned, we split each 1.2 TiB snapshot into 256 files of approximately equal size. We determined that transferring 64 or 128 files concurrently, with a total of 128 or 256 TCP streams, yielded the maximum transfer rate. We achieved an average disk-to-disk transfer rate of 92.4 Gb/s (or 1 PiB in 24 hours and

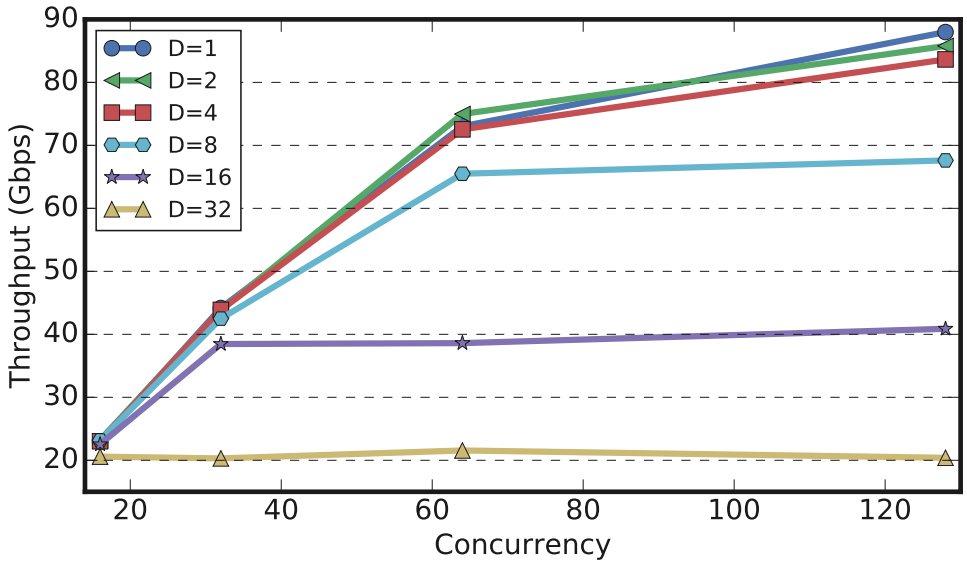

Figure 5: Data transfer performance versus concurrency for different pipeline depth values.

3 minutes): 99.8% of our goal of 1 PiB in 24 hours, when the end-to-end verification of data integrity in Globus is disabled. In contrast, when the end-to-end verification of data integrity in Globus is enabled, we achieved an average transfer rate of only 72 Gb/s (or 1 PiB in 30 hours and 52 minutes).

The Globus approach to checksum verification is motivated by the observations that the 16-bit TCP checksum is inadequate for detecting data corruption during communication [16, 17] and that corruption can occur during file system operations [18]. Globus pipelines the transfer and checksum computation; that is, the checksum computation of the ith file happens in parallel with the transfer of the $(i+1)$th file. Data are read twice at the source storage system (once for transfer and once for checksum) and written once (for transfer) and read once (for checksum) at the destination storage system. Therefore, in order to achieve the desired rate of 93 Gb/s for checksum-enabled transfers, in the absence of checksum failures, 186 Gb/s of read bandwidth from the source storage system and 93 Gb/s write bandwidth and 93 Gb/s of read bandwidth concurrently from the destination storage system are required. If checksum verification failures occur (i.e., one or more files are corrupted during the transfer), even more storage I/O bandwidth, CPU resources, and network bandwidth are required in order to achieve the desired rate. Figure 6 shows the overall transfer throughout, as determined via SNMP network monitoring, and the DTN CPU utilization, when performing transfers using the optimal parameters that we identified.

We see that transfers without integrity checking (marked by dashed line boxes in Figure 6) can sustain rates close to our environment's theoretical bandwidth of 100 Gb/s, with little CPU utilization. If integrity checking is enabled (solid line boxes in Figure 6), however, the CPU utilization increases significantly, and it is hard to get close to the theoretical bandwidth continuously. We note three points: (1) the network is not dedicated to this experiment, and so some network bandwidth was unavoidably consumed by other programs; (2) we used the same optimal tunable parameters (concurrency(128) and parallelism(1)) and the same file size for transfers with and without checksum in order to make sure that the only difference between the two cases is data verification; and (3) there are transfers with non-optimal parameters, performed as part of our explorations, running during other times (outside the boxes) in Figure 6.

4.1. Checksum failures

Globus restarts failed or interrupted transfers from the last checkpoint in order to avoid retransmission costs. In the case of a checksum error, however, it retransmits the entire erroneous file. About 5% of our transfers experienced checksum failure. Such failures can be caused by network corruption, source storage error, or destination storage error. Since the data integrity is verified after the whole file has been transferred to

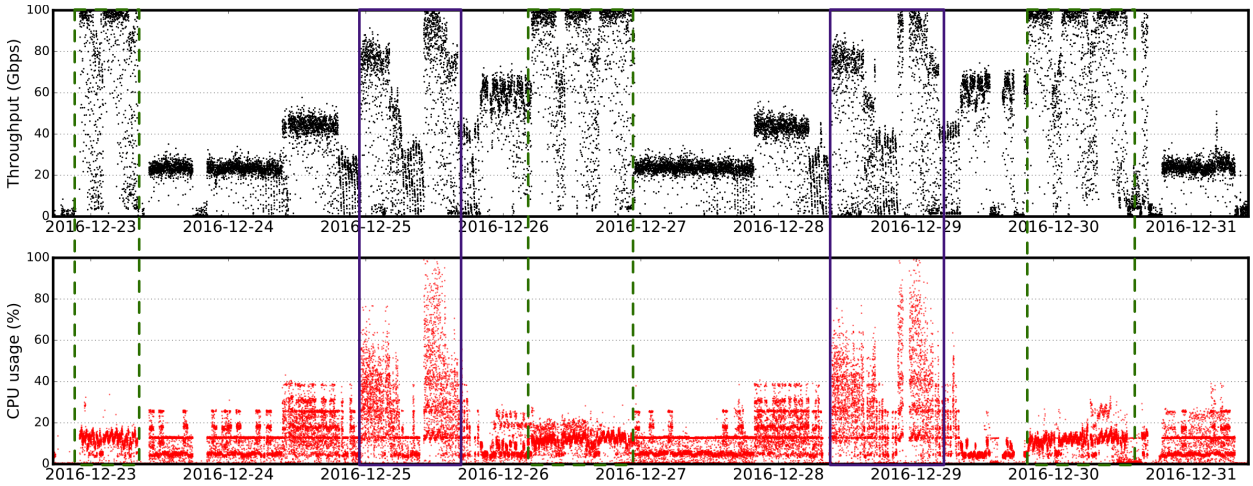

Figure 6: Data transfer throughput vs. DTN CPU usage over eight days. We highlight two periods in which our testing data transfers were occurring with checksum computations (periods delineated by solid line box) and three periods in which our testing transfers were occurring without checksum computations (dashed line boxes). The rest of the periods, i.e., not highlighted, are other users' regular transfers.

the destination, a retransmission must be done if their checksum does not match. For a given transfer, the number of failure represents the number of retransferred files. Obviously the transfer throughput will go down if too many failures occur. We show the transfer throughput versus failures in Figure 7.

Assume that a transfer contains N files, each of x bytes, and there are n checksum verification failures. Thus, n files are retransferred, and the total bytes transferred will then be $(N+n)x$. If we assume that the end-to-end throughout is R_{e2e}, the actual transfer time T_{trs} will be

$$T_{\text{trs}} = \frac{x\,(N+n)}{R_{\text{e2e}}.} \tag{1}$$

Thus, the effective throughout R_{trs} to the transfer users, that is, the time it takes T_{trs} seconds to transfer Nx bytes, will be

$$R_{\text{trs}} = \frac{Nx}{T_{\text{trs}}} = \frac{NR_{\text{e2e}}}{N+n.} \tag{2}$$

We note that transfers in Figure 7 have different concurrency, parallelism, pipeline depth, and average file size; and thus their R_{e2e} are different. If we look only at the transfers with similar concurrency, the shape in Figure 7 fits well with Equation 2.

5. Retrospective analysis: A model-based approach to finding the optimal number of files

The ability to control the file size (and thus number of files) in the dataset is a key flexibility in this use case. Thus, while we used a file size of 4 GB in our experiments, based on limited exploration and intuition, we realized in retrospect that we could have created a model to identify the optimal file size. Here, we present follow-up work in which we develop and apply such a model.

In developing a model of file transfer plus checksum costs, we start with a simple linear model of transfer time for a single file:

$$T_{\text{trs}} = a_{\text{trs}}x + b_{\text{trs}}, \tag{3}$$

where a_{trs} is the unit transfer time, b_{trs} the transfer startup cost, and x the file size. Similarly, we model the time to verify file integrity as

$$T_{\text{ck}} = a_{\text{ck}}x + b_{\text{ck}}, \tag{4}$$

where a_{ck} is the unit checksum time, b_{ck} the checksum startup cost, and x the file size.

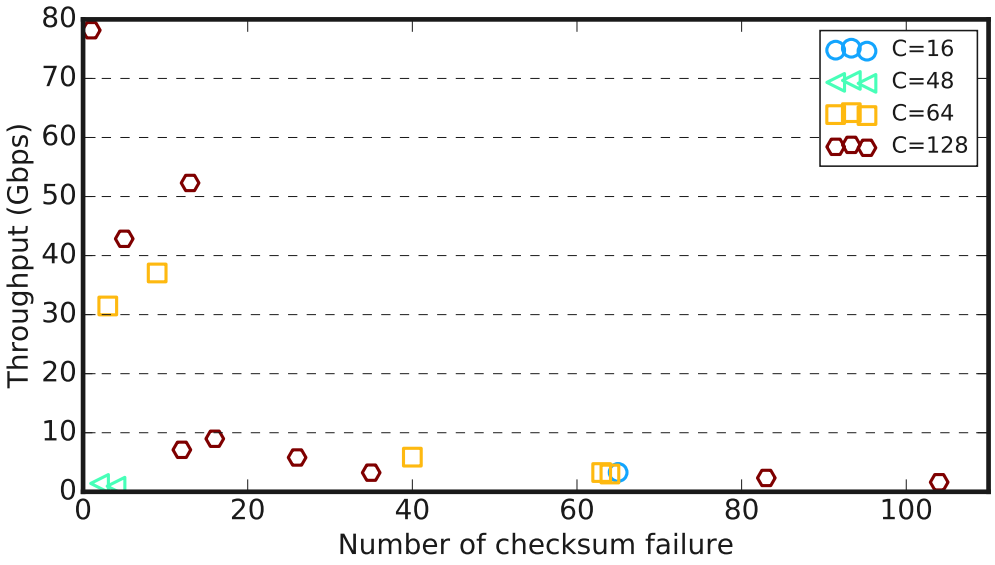

Figure 7: Data transfer performance versus number of checksum failures. Classified by the transfer concurrency (C). Each dot represents one transfer.

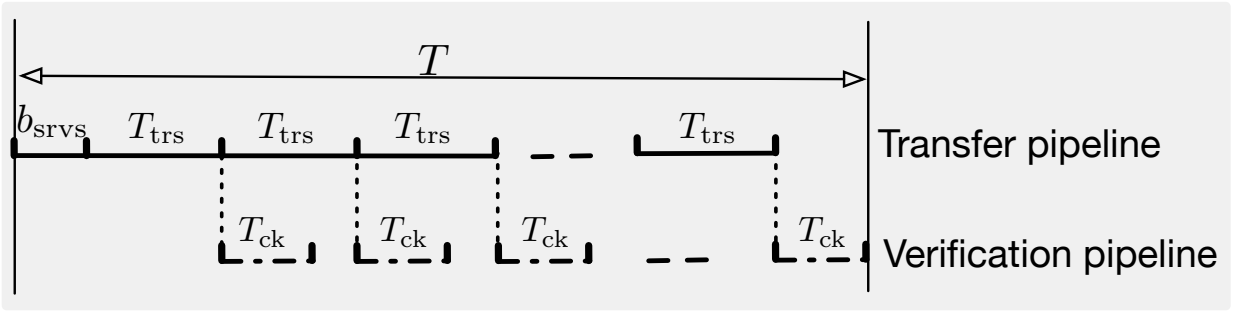

Figure 8: Illustration of file transfer and verification overlap within a single GridFTP process, as modeled by Equation 5. File transfer and associated file verification operations are managed as independent pipelines, connected only in that a file can be verified only after it has been transferred, and verification failure causes a file retransmit.

The time to transfer *and* checksum a file is not the simple sum of these two equations because, as shown in Figure 8, Globus GridFTP pipelines data transfers and associated checksum computations. Note how the data transfer and file integrity verification computations overlap. Thus, assuming no file system contention and that the unit checksum time is less than the unit transfer time (we verified that this is indeed the case in our environment), the total time T to transfer n files with one GridFTP process, each of size x, is the time required for N file transfers and one checksum, namely,

$$T = nT_{\text{trs}} + T_{\text{ck}} + b_{\text{srvs}} = n(xa_{\text{trs}} + b_{\text{trs}}) + xa_{\text{ck}} + b_{\text{ck}} + b_{\text{srvs}}, \tag{5}$$

where b_{srvs} is the transfer service (Globus in this paper) startup cost (e.g., time to establish the Globus control channel). Let us now assume that concurrency $= cc$. S denotes the bytes to be transferred in total; and we equally divide S bytes into N files, where N is perfectly dividable by cc. Thus there are $n = N/cc$ files per concurrent transfer (i.e., per GridFTP process), and each file of size $x = \frac{S}{N}$. The transfer time T to the number of files N will be

$$T(N) = \frac{S}{cc}a_{\text{trs}} + \frac{N}{cc}b_{\text{trs}} + \frac{S}{N}a_{\text{ck}} + b_{\text{ck}} + b_{\text{srvs}}. \tag{6}$$

We use experimental data to estimate the four parameters in Equation 6, a_{trs}, b_{trs}, a_{ck}, and $(b_{\text{ck}} + b_{\text{srvs}})$, in our environment and for different concurrency values, *cc*. Since we previously determined (see Figure 4) that parallelism makes little difference in our low RTT environment, we fixed parallelism at four in these experiments. We note that, for other scenarios with long RTT, the best parallelism should be determined first, e.g., by iteratively exploring parallelism as shown in Figure 4. For each concurrency value, we fixed S=1.2 TiB, used four measured (N, T) points to fit the four parameters, and then used the resulting model to predict performance for other values of N. Figure 9 shows our results. Here, the lines are model predictions, stars are measured values used to fit the model, and other dots are other measured values not used to fit the model.

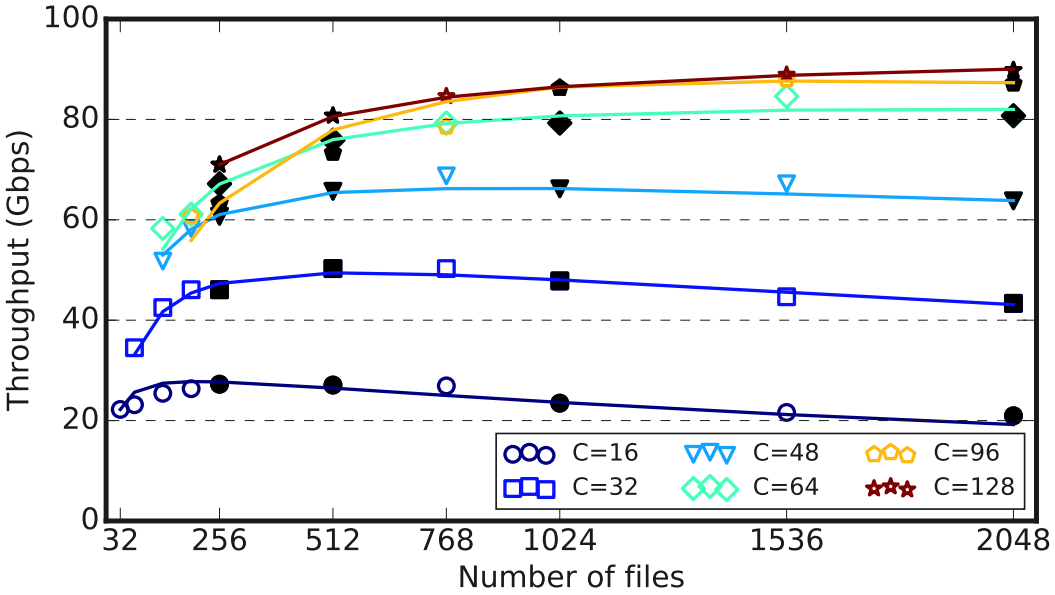

Figure 9: Evaluation of our transfer performance model when transferring a single 1.2 TiB snapshot. Solid markers are points that were used to fit the model parameters shown in Equation 6.

Figure 9 shows the accuracy of the performance model. We see that our model does a good job of predicting throughput as a function of N and *cc*. Since the four model parameters in Equation 6 are independent of source, network, and destination, at least four experiments are needed to fit the model, after which the model can be used to determine the best file size to split a simulation snapshot. We conclude that the optimal file size is around 800 MiB (i.e., split the 1.2 TiB snapshot to 1536–2048 files) and that it can achieve a throughput of 89.7 Gb/s with integrity verification. This throughput represents an increase of 25% compared with that obtained with the ad hoc approach, when we used a file size of 4 GB.

6. Related work

Elephant flows such as those considered here have been known to account for over 90% of the bytes transferred on typical networks [19], making their optimization important.

At the 2009 Supercomputing conference, a multidisciplinary team of researchers from DOE national laboratories and universities demonstrated the seamless, reliable, and rapid transfer of 10 TiB of Earth System Grid data [20] from three sources—the Argonne Leadership Computing Facility, Lawrence Livermore National Laboratory, and National Energy Research Scientific Computing Center. The team achieved a sustained data rate of 15 Gb/s on a 20 Gb/s network provided by DOE's ESnet. More important, their work provided critical feedback on how to deploy, tune, and monitor the middleware used to replicate petascale climate datasets [21]. Their work clearly showed why supercomputer centers need to install dedicated hosts, referred to as data transfer nodes, for wide area transfers [8].

In another SC experiment, this time in 2011, Balman et al. [22] streamed cosmology simulation data over a 100 Gb/s network to a remote visualization system, obtaining an average performance of 85 Gb/s. However, data were communicated with a lossy UDP-based protocol.

Many researchers have studied the impact of parameters such as concurrency and parallelism on data transfer performance [5, 6, 7] and have proposed and evaluated alternative transfer protocols [23, 24, 25] and implementations [26]. Jung et al. [27] proposed a serverless data movement architecture that bypasses data transfer nodes, the filesystem stack, and the host system stack and directly moves data from one disk array controller to another, in order to obtain the highest end-to-end data transfer performance. Newman et al. [28] summarized the next-generation exascale network integrated architecture project that is designed to accomplish new levels of network and computing capabilities in support of global science collaborations through the development of a new class of intelligent, agile networked systems.

Rao et al. [29] 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. Liu et al. [30] similarly studied UDT [31].

Specifically for bulk wide area data transfer, Liu et al. [32] analyzed millions of Globus [3] data transfers involving thousands of DTNs that DTN performance has a nonlinear relationship with load. Liu et al. [33] conducted a systematic examination of a large set of data transfer log data to characterize into transfer characteristics, including the nature of the datasets transferred, achieved throughput, user behavior, and resource usage. Their analysis yields new insights that can help design better data transfer tools, optimize networking and edge resources used for transfers, and improve the performance and experience for end users. Specifically, their analysis show that most of the datasets as well as individual files transferred are very small; data corruption is not negligible for large data transfers; the data transfer nodes utilization is low.

7. Conclusion

We have presented our experiences in transferring one petabyte of science data within one day. We first described the exploration that we performed to identify parameter values that yield maximum performance for Globus transfers. We then discussed our experiences in transferring data while the data are produced by the simulation, both with and without end-to-end integrity verification. We achieved 99.8% of our one petabyte-per-day goal without integrity verification and 78% with integrity verification. We also used a model-based approach to identify the optimal file size for transfers; the results that suggest that we could achieve 97% of our goal *with* integrity verification by choosing the appropriate file size. We believe that our work serves as a useful lesson in the time-constrained transfer of large datasets.

Acknowledgments

We would like to thank the FGCS anonymous reviewers, for the valuable feedback and good questions they brought up. This work was supported in part by the U.S. Department of Energy under contract number DEAC02-06CH11357, National Science Foundation award 1440761 and the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois.

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, W. Liao, HACC: Simulating sky surveys on state-of-the-art supercomputing architectures, New Astronomy 42 (2016) 49–65.

[2] www.globus.org, globus, https://www.globus.org (2018 (accessed January 3, 2018)).

[3] K. Chard, S. Tuecke, I. Foster, Efficient and secure transfer, synchronization, and sharing of big data, IEEE Cloud Computing 1 (3) (2014) 46–55.
[4] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, I. Foster, Swift: A language for distributed parallel scripting, Parallel Computing 37 (9) (2011) 633–652.
[5] T. J. Hacker, B. D. Athey, B. Noble, The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network, in: Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM, IEEE, 2001, pp. 10–pp.
[6] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, The Globus striped GridFTP framework and server, in: ACM/IEEE Conference on Supercomputing, IEEE Computer Society, 2005, p. 54.
[7] E. Yildirim, E. Arslan, J. Kim, T. Kosar, Application-level optimization of big data transfers through pipelining, parallelism and concurrency, IEEE Transactions on Cloud Computing 4 (1) (2016) 63–75.
[8] E. Dart, L. Rotman, B. Tierney, M. Hester, J. Zurawski, The Science DMZ: A network design pattern for data-intensive science, Scientific Programming 22 (2) (2014) 173–185.
[9] Z. Liu, R. Kettimuthu, S. Leyffer, P. Palkar, I. Foster, A mathematical programming- and simulation-based framework to evaluate cyberinfrastructure design choices, in: 2017 IEEE 13th International Conference on e-Science, 2017, pp. 148–157. doi:10.1109/eScience.2017.27.
[10] Z. Liu, R. Kettimuthu, I. Foster, P. H. Beckman, Towards a smart data transfer node, in: 4th International Workshop on Innovating the Network for Data Intensive Science, 2017, p. 10.
[11] BBCP, http://www.slac.stanford.edu/~abh/bbcp/.
[12] FDT, FDT - Fast Data Transfer, http://monalisa.cern.ch/FDT/ (2018 (accessed January 3, 2018)).
[13] P. Fuhrmann, V. Gülzow, dCache, storage system for the future, in: European Conference on Parallel Processing, Springer, 2006, pp. 1106–1113.
[14] CERN, FTS3: Robust, simplified and high-performance data movement service for WLCG, http://fts3-service.web.cern.ch (2018 (accessed January 3, 2018)).
[15] B. W. Settlemyer, J. D. Dobson, S. W. Hodson, J. A. Kuehn, S. W. Poole, T. M. Ruwart, A technique for moving large data sets over high-performance long distance networks, in: 27th Symp. on Mass Storage Systems and Technologies, 2011, pp. 1–6. doi:10.1109/MSST.2011.5937236.
[16] V. Paxson, End-to-end internet packet dynamics, IEEE/ACM Transactions on Networking 7 (3) (1999) 277–292.
[17] J. Stone, C. Partridge, When the CRC and TCP checksum disagree, in: ACM SIGCOMM Computer Communication Review, Vol. 30, ACM, 2000, pp. 309–319.
[18] L. N. Bairavasundaram, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, G. R. Goodson, B. Schroeder, An analysis of data corruption in the storage stack, ACM Transactions on Storage 4 (3) (2008) 8.
[19] K. Lan, J. Heidemann, A measurement study of correlations of internet flow characteristics, Computer Networks 50 (1) (2006) 46–62.
[20] D. N. Williams, R. Drach, R. Ananthakrishnan, I. T. Foster, D. Fraser, F. Siebenlist, D. E. Bernholdt, M. Chen, J. Schwidder, S. Bharathi, A. L. Chervenak, R. Schuler, M. Su, D. Brown, L. Cinquini, P. Fox, J. Garcia, D. E. Middleton, W. G. Strand, N. Wilhelmi, S. Hankin, R. Schweitzer, P. Jones, A. Shoshani, A. Sim, The Earth System Grid: Enabling access to multimodel climate simulation data, Bulletin of the American Meteorological Society 90 (2) (2009) 195–205. doi:10.1175/2008BAMS2459.1.
[21] R. Kettimuthu, A. Sim, D. Gunter, B. Allcock, P.-T. Bremer, J. Bresnahan, A. Cherry, L. Childers, E. Dart, I. Foster, K. Harms, J. Hick, J. Lee, M. Link, J. Long, K. Miller, V. Natarajan, V. Pascucci, K. Raffenetti, D. Ressman, D. Williams, L. Wilson, L. Winkler, Lessons learned from moving Earth System Grid data sets over a 20 Gbps wide-area network, in: 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, ACM, New York, NY, USA, 2010, pp. 316–319. doi:10.1145/1851476.1851519.
[22] M. Balman, E. Pouyoul, Y. Yao, E. Bethel, B. Loring, M. Prabhat, J. Shalf, A. Sim, B. L. Tierney, Experiences with 100Gbps network applications, in: 5th International Workshop on Data-Intensive Distributed Computing, ACM, 2012, pp. 33–42.
[23] E. Kissel, M. Swany, B. Tierney, E. Pouyoul, Efficient wide area data transfer protocols for 100 Gbps networks and beyond, in: 3rd International Workshop on Network-Aware Data Management, ACM, 2013, p. 3.
[24] D. X. Wei, C. Jin, S. H. Low, S. Hegde, FAST TCP: Motivation, architecture, algorithms, performance, IEEE/ACM Transactions on Networking 14 (6) (2006) 1246–1259.
[25] L. Zhang, W. Wu, P. DeMar, E. Pouyoul, mdtmFTP and its evaluation on ESNET SDN testbed, Future Generation Computer Systems.
[26] H. Bullot, R. Les Cottrell, R. Hughes-Jones, Evaluation of advanced TCP stacks on fast long-distance production networks, Journal of Grid Computing 1 (4) (2003) 345–359.
[27] E. S. Jung, R. Kettimuthu, High-performance serverless data transfer over wide-area networks, in: 2015 IEEE International Parallel and Distributed Processing Symposium Workshop, 2015, pp. 557–564. doi:10.1109/IPDPSW.2015.69.
[28] H. Newman, M. Spiropulu, J. Balcas, D. Kcira, I. Legrand, A. Mughal, J. Vlimant, R. Voicu, Next-generation exascale network integrated architecture for global science, Journal of Optical Communications and Networking 9 (2) (2017) A162–A169.
[29] N. S. Rao, Q. Liu, S. Sen, D. Towlsey, G. Vardoyan, R. Kettimuthu, 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.
[30] Q. Liu, N. S. Rao, C. Q. Wu, D. Yun, R. Kettimuthu, I. T. Foster, Measurement-based performance profiles and dynamics of UDT over dedicated connections, in: 24th International Conference on Network Protocols, IEEE, 2016, pp. 1–10.

[31] Y. Gu, R. L. Grossman, UDT: UDP-based data transfer for high-speed wide area networks, Computer Networks 51 (7) (2007) 1777–1799.

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

[33] Z. Liu, R. Kettimuthu, I. Foster, N. S. Rao, Cross-geography scientific data transfer trends and user behavior patterns, in: 27th ACM Symposium on High-Performance Parallel and Distributed Computing, HPDC '18, ACM, New York, NY, USA, 2018, p. 12.