Towards Optimizing Large-Scale Data Transfers with End-to-End Integrity Verification

Si Liu*, Eun-Sung Jung†, Rajkumar Kettimuthu‡, Xian-He Sun*, Michael Papka§
*Department of Computer Science, Illinois Institute of Technology
†Computer and Information Communication Department, Hongik University
‡Mathematics and Computer Science Division, Argonne National Laboratory
§Argonne Leadership Computing Facility, Argonne National Laboratory
Email: *sliu89@hawk.iit.edu, sun@iit.edu †ejung@hongik.ac.kr, ‡kettimut@anl.gov, §papka@anl.gov

***Abstract*—The scale of scientific data generated by experimental facilities and simulations on high-performance computing facilities has been growing rapidly. In many cases, this data needs to be transferred rapidly and reliably to remote facilities for storage, analysis, sharing etc. At the same time, users want to verify the integrity of the data by doing a checksum after the data has been written to disk at the destination, to ensure the file has not been corrupted, for example due to network or storage data corruption, software bugs or human error. This end-to-end integrity verification creates additional overhead (extra disk I/O and more computation) and increases the overall data transfer time. In this paper, we evaluate strategies to maximize the overlap between data transfer and checksum computation. More specifically, we evaluate file-level and block-level (with various block sizes) pipelining to overlap data transfer and checksum computation. We evaluate these pipelining approaches in the context of GridFTP, a widely used protocol for science data transfers. We conducted both theoretical analysis and real experiments to evaluate our methods. The results show that block-level pipelining is an effective method in maximizing the overlap between data transfer and checksum computation and can improve the overall data transfer time with end-to-end integrity verification by up to 70% compared to the sequential execution of transfer and checksum, and by up to 60% compared to file-level pipelining.**

Keywords *- High-performance data transfer; Data integrity; Pipelining*

I. Introduction

The scale of scientific data generated by experimental facilities and simulations on high-performance computing environments has been growing rapidly. For example, in cosmology Dark Energy Survey (DES) telescope in Chile captures TBs of data per night, another cosmology project Square Kilometer Array [1] will generate an exabyte every 13 days when it becomes operational in 2024. DOE light source facilities generate tens of TBs of data per day now and are poised to increased by two orders of magnitude in the next few years. The Compact Muon Solenoid (CMS) experiment is one of the four detectors located at the LHC [2]. It is designed to record particle interactions occurring at its center. Every year, CMS records and simulates 6 petabytes of proton-proton collision data to be processed and analyzed.

These large datasets are often transferred over wide-area networks for multiple purposes, such as storage, analysis and visualization. When moving large quantities of data across end-to-end storage system-to-storage system paths, it is essential to do end-to-end checksum verification. Even though some of the components in the end-to-end path implement their own data integrity check, it is insufficient. For example, TCP in the network communications implements the TCP checksum [3] and storage controllers in data storage systems implement their own data integrity methods [4]. However, these are insufficient for two reasons: 1) it does not cover the complete end-to-end path of the data transfer and 2) the probability of integrity failure increases exponentially with multiple components (a transfer involving 10 components each with their own integrity check that captures 99% data corruption would result in 10% $(1 - .99^{10})$ undetected data corruption).

In addition, J. Stone et. al [5] showed through extensive real experiments that TCP checksum is not enough to guarantee the end-to-end data integrity. A 16-bit checksum means that 1 in 65,536 bad packets will be erroneously accepted as correct. According to [6], around 1 in 5,000 Internet data packets is corrupted in transit. Thus, around 1 in every 300 Million (65K*5K) packets is accepted with corruption. It has been reported that an average of 40 errors per million transfers is detected on data transferred by the D0 experiment [7]. Projects such as DES require verification of checksums as part of their regular data movement process in order to catch file corruption due to software bugs or human error.

While end-to-end data integrity check is essential in big data transfer, it does not come for free. It creates additional overhead in terms of disk I/O and computation, which increases the overall data transfer time. Based on the tests we conducted with Globus [8] and the analysis of GridFTP transfer logs from some sites indicate that the checksum overhead can be anywhere between 30% and 100%. In this work, we evaluate file-level and block-level (with various block sizes) pipelining strategies to overlap data transfer and checksum computation. We conduct both theoretical analysis and experiments on real testbeds to evaluate these strategies. File-level pipelining is employed in production data transfer mechanisms such as Globus. To the best of our knowledge, block-level pipelining in an end-to-end fashion is not employed in practice for large-scale file transfers. Our results show that block-level pipelining is an effective method in maximizing the overlap between data transfer and checksum computation. Block-level pipelining can improve the overall data transfer time with end-to-end integrity verification by up to 70% compared to the sequential execution of transfer and checksum, and by up to 60% compared to file-level pipelining, for synthetic datasets. For a real science dataset, the improvement is up to 57% compared to the sequential execution and 47% compared to file-level pipelining.

The rest of the paper is organized as follows. In Section II, we summarize the related work on high-performance data transfer and associated data integrity issues. In Section III, we describe pipeline approaches to optimize high-performance data transfer with end-to-end data integrity check. In Section IV, we present experimental results on real testbeds to evaluate the effectiveness of the pipeline approaches. We conclude with the brief summary of the work and future work in Section V.

II. Related Work

Many tools have been developed for file transfers – GridFTP [9], Globus file transfer [8], bbcp [10], FDT [11], XDD [12] to name a few. A number of approaches have been proposed to optimize large-scale wide-area data transfers. [13] proposes an algorithm that dynamically schedules a batch of data transfer requests to minimize the overall transfer time. The use of multiple TCP streams and concurrent file transfers is often required in order to achieve file transfer rates close to network speeds [14] [15]. Kettimuthu et. al incorporated on-the-fly checksum capabilities in GridFTP but it is not true end-to-end in the sense that it does not account for any data corruption in the path from the host to the storage system on the data receiver. To address the need for end-to-end checksum as well as to address the limitations of 16-bit TCP checksum, Globus transfer service incorporated an additional 128-bit checksum computation (reduces the number of undetected bad packets to one in $2 * 10^{13}$) by reading the file at the destination after it is written to the disk. Globus supports file-level pipelining of transfer and checksum by overlapping the checksum computation of a previously transfered file with the transfer of another file for multi-file transfers. We evaluate this approach in our work in addition to the block-level pipelining of transfer and checksum computation, which is not currently employed for production science data transfers, to the best of our knowledge.

III. Pipelining data transfer and end-to-end data integrity check

In this section, we describe our methods for high-performance end-to-end data integrity check. First, we introduce the main ”pipelining” strategy in our methods, which are classified into file-level pipelining and block-level pipelining. Besides, the analytical modeling for performance analysis is also established and illustrated in this part. Second, we also explore potentials to enhance the block-level pipelining.

A. Pipelining

Pipelining is a useful parallelizing technique to improve the repetitive tasks composed of multiple steps. We make use of this pipelining technique to achieve high-performance data transfer with data integrity check, which is composed of a data transfer operation and a data integrity check operation. Figure 1 shows the illustrative example of pipelining data transfer and data integrity check. **T** represents data transfer. **C** represents data integrity check.

File-level pipelining vs. block-level pipelining. File-level pipelining overlaps a file transfer and a file integrity check while block-level pipelining overlaps a block (whose size is less than the average file size in a dataset) transfer and block data integrity check. Theoretically, the pipelined operations work best when all the operations take the same time. In other words, the performance of the pipelined operations are

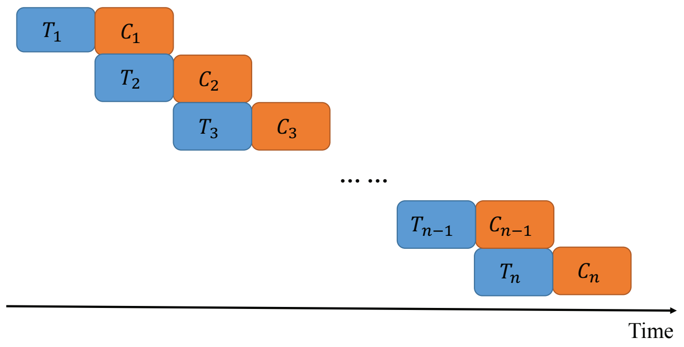

Fig. 1: Pipelining data transfer and data integrity check

bottlenecked by the longest operation. Considering the two operations, i.e. data transfer and data integrity check, the executions times vary depending on platforms. For example, the data transfer time may be longer than the data integrity check time in case of slow network connections, and the data integrity check time may be longer in case of high-speed network connections and low-end (or highly loaded) CPU and/or storage-system. Block-level pipelining can reduce the gap between the data transfer time and data checksum time because file-level pipelining overlaps two operations for two different files. Suppose that 10MB file transfer is overlapped with the data checksum for the previous file of size 10GB or vice versa. The gap between transfer time and checksum time could be huge in this case. This problem can be resolved in block-level pipelining where the gap (e.g. difference of data transfer time for 10MB and data checksum time for 10MB if the block size is 10MB.) is always constant.

Analytical Modeling. We analyze the performance of block-level pipelining using data transfer time and data checksum time. We can model the performance of the block-level pipelining for two cases: 1) when data transfer time is longer than checksum time (Transfer-Dominant Case), and 2) when data checksum time is longer than checksum time (Checksum-Dominant Case). Based on tests, we found that both transfer time and checksum time (md5sum) are a linear function of data size in a relatively contention free environment.

Since it is hard to analytically model the performance of a dataset consisting of multiple random files, we generate two extreme cases of synthetic datasets having distintive file size patterns, a dataset consisting of twenty 10GB files (20-10GB) and a dataset consisting of ten repetitions of a 10GB file and a 500MB file (10GB-500MB) where a 10GB file transfer precedes 500MB file transfer. We use these two datasets, i.e. 20-10GB and 10GB-500MB, to perform analytical modeling. We also experimentally evaluate the performance for these datasets and verify that the results are in line with the analytical models.

For mathematical analysis, we define $\mathbf{t}$ and $\mathbf{c}$ as follows.

- $\mathbf{t}$: Transfer time of 500MB data
- $\mathbf{c}$: Checksum time of 500MB data

For our experiments, we deliberately chose two different testbeds one in which the transfer time is dominant and one in which the checksum computation time is dominant. So, we have separate analytical models for these two cases as

TABLE I: Comparison of analytical performance models of 1) block-level pipeline, 2) file-level pipeline, and 3) sequential file transfer (existing method for baseline)

Case	Dataset	Block-level Pipeline				File-level Pipeline	File Sequential
		100MB	500MB	1GB	2GB		
Transfer-Dominant	20-10GB	$400 \times t + 1/5 \times c$	$400 \times t + c$	$400 \times t + 2 \times c$	$400 \times t + 4 \times c$	$400 \times t + 20 \times c$	$400 \times (t + c)$
$t > c$	10GB-500MB	$210 \times t + c$	$210 \times t + c$	$210 \times t + c$	$210 \times t + c$	$200 \times t + 201 \times c$	$210 \times (t + c)$
Checksum-Dominant	20-10GB	$400 \times c + 1/5 \times t$	$400 \times c + t$	$400 \times c + 2 \times t$	$400 \times c + 4 \times t$	$400 \times c + 20 \times t$	$400 \times (c + t)$
$t < c$	10GB-500MB	$208 \times c + 51/5 \times t$	$210 \times c + t$	$210 \times c + 2 \times t$	$201 \times c + 40 \times t$	$200 \times t + 201 \times c$	$210 \times (c + t)$

summarized in Table I. Note that we transfer the whole file of 500MB files in case of 10GB-500MB dataset even with 100MB block size while transferring 10GB files in 100MB blocks. Each model is generated based on the number of pipeline stages in each case and execution time for each stage, both of which are decided by the number of blocks and the times of each block over 500MB.

B. Enhancing Block-level Pipelining

Judging from the analytical performance modeling of the block-level pipelining and general pipelining behaviors, the best performance can be achieved when the data transfer time is close to the data checksum time. Hence, we explore opportunities of making the time of the two operations as close as possible.

In Checksum-Dominant case, we can make the time of the two operations close by reducing the data checksum time. We use multiple threads (and cores) to compute checksums of multiple parts of a block in parallel. In Transfer-Dominant case, even though one could reduce the transfer time by compressing the data size of a block, it is not straightforward as the compression ratio depends on the dataset and algorithm used. In this work, we reduce the gap between block transfer time and block checksum time for Checksum-Dominant cases by parallelizing the checksum computation. Use of compression to reduce the gap between block transfer time and block checksum time for Transfer-Dominant cases could be a subject for future work.

IV. Experimental Evaluation

In this section, we describe our testbeds for real data transfer tests, and present experimental results followed by in-depth discussion on them. We first describe two experimental testbeds, Cooley and Rains, with different configurations representing Checksum-Dominant Case and Transfer-Dominant Case, respectively. We then present our evaluation methodology and the experimental results.

A. Experimental testbeds

We evaluate 3 schemes:

- **Sequential (baseline)**, where the file transfer and checksum computation are completely serialized.
- **File-level Pipeline**, where the checksum computation of file X is done in parallel with the transfer of file X+1 (for all X < M, where M is the number of files in the dataset).
- **Block-level Pipeline**, where the checksum computation of block Y of file X is done in parallel with the transfer of block Y+1 of file X (for all Y < N, where N is the number of blocks in file X) and the checksum computation of block N of file X is done in parallel with the block 1 of file X+1.

We evaluated block-level pipeline for various block sizes. We conducted experiments on two different clusters at Argonne National Laboratory – *Cooley* at the Argonne Leadership Computing Facility (ALCF) and *Rains* at the Joint Laboratory for System Evaluation (JLSE). *Cooley* is Checksum-Dominant and *Rains* is Transfer-Dominant. Both clusters have parallel/shared file systems, i.e. GPFS, as storage systems. We picked two nodes (one sender and one receiver) on each cluster to run our tests.

1) Cooley (Checksum-Dominant Case)

   Cooley [16] is an analysis and visualization cluster at ALCF

   A node in Cooley testbed, a cluster of 126 nodes, has the following hardware configuration.

   - Architecture: Intel Haswell
   - Processors: Two 2.4 GHz Intel Haswell E5-2620 v3 processors per node (6 cores per CPU, 12 cores total)
   - Memory/node: 384GB RAM per node, 24 GB GPU RAM per node (12 GB per GPU)
   - Network: 10Gbps
   - FDR Infiniband interconnect for GPFS

2) Rains (Transfer-Dominant Case)

   A node in Rains testbed, a cluster of 16 nodes, has the following hardware configuration.

   - Architecture: AMD Opteron
   - Processors: Four AMD Opteron 2216 processors per node (2 cores per CPU, 8 cores total)
   - Memory/node: 8GB
   - Network: 1Gbps
   - 20 Gb DDR InfiniBand interconnect for GPFS

B. Evaluation methodology

We employed GridFTP as a data transfer tool for our experiments. Our method will apply to other data transfer tools without loss of generality. Globus transfer service [8] and *globus-url-copy* are the commonly used clients for GridFTP. Both of them support only file-level checksum. We use the later for our tests. Since it supported only file-level checksum, We computed the checksums for all the 3 schemes by running the Linux system command *md5sum* in a separate thread. We verified that the performance of built-in checksum in *globus-url-copy* is close to that of Linux system command *md5sum*.

Experimental datasets. We generated three synthetic datasets to evaluate the performance of different data transfer methods.

1) 10GB-500MB dataset: Dataset with 10 10GB files and 10 500MB files, 105GB in total.
2) 20-10GB dataset: Dataset with 20 the same size of 10GB files, 200GB in total.
3) Real dataset: Dataset generated based on the distribution of Intergovernmental Panel on Climate Changes (IPCC) Coupled Model Intercomparison Project 3 (CMIP-3) dataset, 174GB in total.

The first two datasets, 10GB-500MB and 20-10GB, represent the extreme cases to show the effects of pipelining methods. The real dataset is used to evaluate the performance of pipelining methods for real-world use cases. 10GB-500MB dataset is composed of files with only two sizes, one is large, the other one is small. This kind of dataset will get the most benefit from block-level pipeline. 20-10GB dataset is composed of files of the same size where the block-level pipeline should have little benefit over file-level pipeline. Real dataset is composed of files with sizes ranging from Megabytes to Gigabytes. We expect the performance of block-level pipeline for this case to be in between the performance achieved by block-level pipeline for 10GB-500MB dataset and that for 20-10GB dataset.

For all experiments, we measured the performance of sequential file transfer, file-level pipeline transfer, and block-level pipeline transfer with block sizes of 100MB, 500MB, 1GB and 2GB. For the real dataset, we used block sizes of 50MB, 100MB, 500MB and 1GB. The reason is that file size in the real dataset varied a lot from 10MB to 2.1GB with 1/3 of files less than 50MB and only a few files are larger than 2GB.

We used the partial file transfer feature in GridFTP to do block-level transfers, which introduces startup (connection setup, additional protocol, and TCP ramp-up) overhead for transferring each block. In practice, block-level pipeline will be done inside the data transfer tool and will not result in a separate startup overhead for each block. Thus, we remove the additional startup overhead for block-level transfers. In order to do so, we measured the startup overhead on two testbeds, Cooley and Rains, based on the following methodology:

T1: Time for transferring 20 blocks of the same size, e.g. 500MB. **T2**: Time for transferring one file with size of 20 times of the block size, (e.g. 10GB= 500MB $\times$ 20).

$$\text{Startup overhead} = (T1 - T2)/19 \qquad (1)$$

C. Block-level pipelining

1) Results on Cooley (Checksum-Dominant Case, $\mathbf{t} < \mathbf{c}$): On Cooley testbed, we measured $\mathbf{t} \approx 0.5$ secs, $\mathbf{c} \approx 1$ secs. Analytical performance can be obtained as follows. After substituting $\mathbf{t}$ with value 0.5 and $\mathbf{c}$ with value 1 in the formulas in Table I, we can calculate the approximate performance gain of block-level pipelining over file-level pipelining and file sequential transfers. For dataset 20-10GB, the maximum performance gain over file-level pipelining is around 10 seconds; For dataset 10GB-500MB, the maximum performance gain over file-level pipelining is around 80 seconds. Compared to file sequential transfers, the performance gain is around 200 seconds and 100 seconds respectively for 20-10GB and 10GB-500MB datasets.

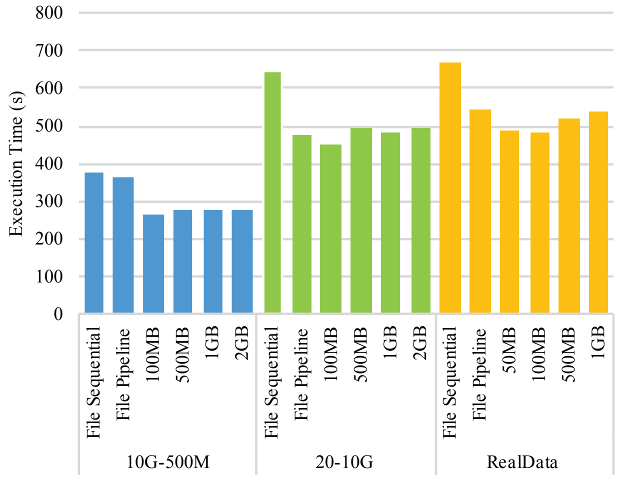

Fig. 2: Performance comparison of Sequential, File-level pipeline, and Block-level pipeline (for different block sizes) on Cooley

Figure 2 shows the performance of sequential, file-level pipeline and block-level pipeline (for different block sizes) for different datasets on Cooley. The results are consistent with the theoretical analysis: For dataset 10GB-500MB, the performance gain of block level pipeline over file-level transfers is around 100 seconds (or $\sim$30%); For dataset 20-10GB, there is little performance gain (less than 10 seconds); For real dataset, the performance gain percentage ($\sim$16%) is in between the other two datasets. The block size of 100MB is the best for all three datasets. And the performance seems to degrade with increasing block size with the exception of 1GB performing better than 500MB for 20-10GB dataset. We suspect that the reason for this is: the transfer time is not perfectly linear with respect to the dataset size (and same is the case for checksum computation time too) and the overlap between the transfer time and checksum computation time is best for the block size of 100MB.

2) Results on Rains (Transfer-Dominant Case, $\mathbf{t} > \mathbf{c}$): On Rains testbed, we measured $\mathbf{t} \approx 7$ secs, $\mathbf{c} \approx 1$ secs. Analytical performance can be obtained as follows. After substituting $\mathbf{t}$ with value 7 and $\mathbf{c}$ with value 1 in the formulas in Table I, we can calculate the approximate performance gain of block-level pipelining over file-level pipelining and file sequential transfers. For dataset 20-10GB, the maximum performance gain over file-level pipelining is around 20 seconds; For dataset 10GB-500MB, the maximum performance gain over file-level pipelining is around 200 seconds. Compared to file sequential transfers, the performance gain is around 400 seconds and 200 seconds respectively for 20-10GB and 10GB-500MB datasets.

Figure 3 shows the performance of sequential, file-level

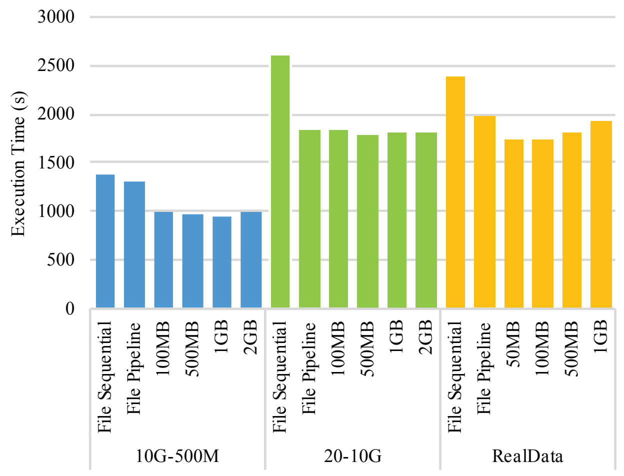

Fig. 3: Performance comparison of Sequential, File-level pipeline, and block-level pipeline (for different block sizes) on Rains

pipeline and block-level pipeline (for different block sizes) for different datasets on Rains. The results are consistent with the theoretical analysis: For dataset 10GB-500MB, the performance gain of block-level pipeline over file-level pipeline is around 280 seconds (∼22%); For dataset 20-10GB, there is little performance gain (less than 10 seconds); For real dataset, the performance gain percentage (∼14%) is in between the other two datasets. With a significant gap between the block transfer and block checksum times, we do not expect any significant difference in performance between different block sizes. From Figure 3, it can observed that this is in fact the case for 10GB-500MB and 20-10GB datasets but for the real dataset, the performance of 1G block size is worse than the others. It requires further investigation.

3) Perfect Pipeline: The results of the previous experiments have shown that block-level pipeline transfer is an effective method to achieve high-performance transfer with data integrity check. But we could not achieve perfect pipeline yet due to the difference between block transfer time and block checksum computation time. For instance, the transfer time is half of the checksum time on Cooley.

Now, we want to see the impact of the perfect pipeline. In order to achieve perfect pipeline between block transfer and block checksum computation, we parallelized the checksum computation using 2 threads (cores). We have two checksum threads – one responsible for computing the checksum of first half of a block and the other responsible for computing the checksum of second half of a block. Figure 4 shows performance comparisons between 1-Checksum-Thread case and 2-Checksum-Thread case. It shows that the execution time of 2-Checksum-Thread is almost half of the 1-Checksum-Thread in most block-level cases in each dataset except for the smallest block size (where the overhead of using 2 threads possibly dominates). In 2-Checksum-Thread case, 500MB block size is clearly the best for all three datasets (Please note that 500MB block size is last but one bar for the real dataset and last but two bar for the other two datasets). It is because the number of threads for checksum computation is picked in such a way to get complete overlap for the 500MB block size. And the 500MB block size gets almost a linear speedup with 2 threads. Others block sizes get significant performance improvement with 2 threads for checksum computation as checksum computation dominates the transfer time in the Cooley testbed.

Unlike the Cooley testbed, the Rains testbed is dominated by the transfer time. Based on the results on Cooley, we believe that similar performance gain can be achieved if we can reduce the transfer time (by compressing the data in advance of transferring over the networks) to match the checksum computation time. But the performance improvement is dependent on the compressibility of the dataset, available CPU resources etc. If the data is readily compressible and if there are enough CPU resources are available for compression, similar performance gains as in case of the cooley testbed should be achievable.

V. Conclusion and Future Work

The work presented in this paper is a primary summary of our current work on block-level pipelining to overlap data transfer and checksum computation. Based on the theoretical analysis and experimental results, we concluded that block-level pipeline is an effective approach to optimize data transfers with end-to-end integrity checking. We showed that block-level pipeline can improve the overall data transfer time with end-to-end integrity verification by up to 57% compared to the sequential execution of transfer and checksum, and by up to 47% compared to file-level pipelining for a real science dataset. The results obtained provide motivation to explore more optimized methods based on current work. As we can see from the above experimental results, the performance of block-level pipelining varies for different block sizes. In addition, as the 2-Checksum-Thread experiment results indicate, highest performance gains can be achieved when the transfer time and checksum time match (or are very close to each other). We plan to study how to determine appropriate block size, data integrity algorithm (Besides MD5, other data integrity algorithms such as CRC [17], adler32 [18] are used for wide-area data transfers), data compression methods, number of threads to use for checksum computation and/or compression based on the environment and dataset. Some of these choices (e.g., block size) may have to be varied dynamically during the transfer.

Acknowledgments

This work was supported in part by the U.S. Department of Energy under contract number DEAC02-06CH11357 and the National Science Foundation, under grants ACI-1440761 and ACI-1440797.

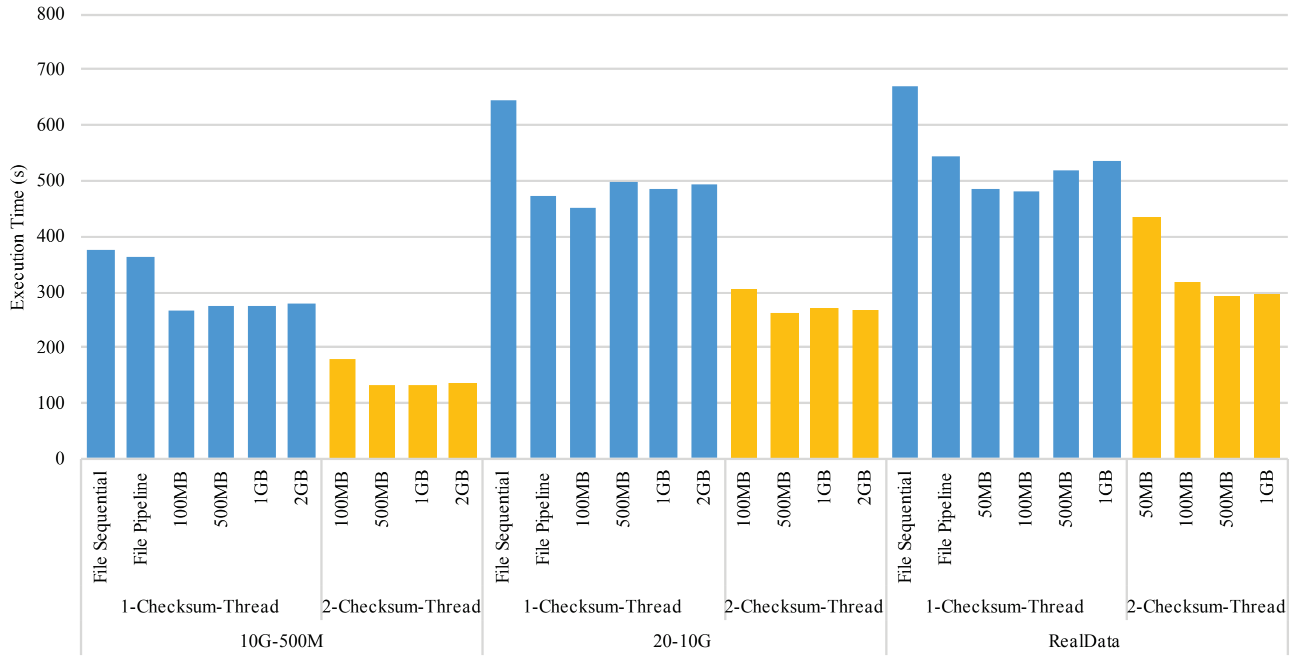

Fig. 4: Comparison of the performance of 1-Checksum-Thread and 2-Checksum-Thread on Cooley

research in accordance with the DOE Public Access Plan. http://energy.gov/downloads/doe-public-access-plan.

References

[1] "Square Kilometre Array." [Online]. Available: https://www.scribd.com/doc/125147649/Ultimate-Big-Data-Challenge#page=1

[2] N. Magini, N. Ratnikova, P. Rossman, A. Snchez-Hernndez, and T. Wildish, "Distributed data transfers in CMS," *Journal of Physics: Conference Series*, vol. 331, no. 4, p. 042036, Dec. 2011. [Online]. Available: http://stacks.iop.org/1742-6596/331/i=4/a=042036?key=crossref.6a079fb4c1305adf13bca4d2fb1a8d3a

[3] T. Maxino and P. Koopman, "The Effectiveness of Checksums for Embedded Control Networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 1, pp. 59–72, Jan. 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4358707

[4] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, "The new ext4 filesystem: current status and future plans," in *Proceedings of the Linux symposium*, vol. 2. Citeseer, 2007, pp. 21–33. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.111.798&rep=rep1&type=pdf#page=21

[5] J. Stone and C. Partridge, "When the CRC and TCP checksum disagree," in *ACM SIGCOMM computer communication review*, vol. 30. ACM, 2000, pp. 309–319.

[6] V. Paxson, "End-to-end internet packet dynamics," in *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '97. New York, NY, USA: ACM, 1997, pp. 139–152. [Online]. Available: http://doi.acm.org/10.1145/263105.263155

[7] "Enabling PETASCALE Data Movement and Analysis." [Online]. Available: http://www.scidacreview.org/0905/html/cedps.html

[8] 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*, vol. 55, no. 2, pp. 81–88, 2012. [Online]. Available: http://doi.acm.org/10.1145/2076450.2076468

[9] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The Globus Striped GridFTP Framework and Server," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, ser. SC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 54–64. [Online]. Available: http://dx.doi.org/10.1109/SC.2005.72

[10] "bbcp." [Online]. Available: http://www.slac.stanford.edu/~abh/bbcp/

[11] "Fast Data Transfer." [Online]. Available: http://monalisa.cern.ch/FDT/

[12] B. W. Settlemyer, J. D. Dobson, S. W. Hodson, J. A. Kuehn, S. W. Poole, and T. M. Ruwart, "A technique for moving large data sets over high-performance long distance networks," *Mass Storage Systems and Technologies, IEEE / NASA Goddard Conference on*, vol. 0, pp. 1–6, 2011.

[13] G. Khanna, U. Catalyurek, T. Kurc, R. Kettimuthu, P. Sadayappan, and J. Saltz, "A dynamic scheduling approach for coordinated wide-area data transfers using gridftp," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE, 2008, pp. 1–12. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4536325

[14] E. Yildirim and T. Kosar, "End-to-End Data-Flow Parallelism for Throughput Optimization in High-Speed Networks," *Journal of Grid Computing*, vol. 10, no. 3, pp. 395–418, Aug. 2012. [Online]. Available: http://link.springer.com/article/10.1007/s10723-012-9220-9

[15] W. E. Johnston, E. Dart, M. Ernst, and B. Tierney, *Enabling high throughput in widely distributed data management and analysis systems: Lessons from the LHC*.

[16] "Cooley." [Online]. Available: http://www.alcf.anl.gov/user-guides/cooley

[17] J. S. Sobolewski, "Cyclic redundancy check," 2003.

[18] P. Deutsch and J.-L. Gailly, "Zlib compressed data format specification version 3.3," Tech. Rep., 1996.