PAPER

High-Performance End-to-End Integrity Verification on Big Data Transfer*

Eun-Sung JUNG[†], *Member*, Si LIU^{††}, Rajkumar KETTIMUTHU^{†††}, and Sungwook CHUNG^{††††a)}, *Nonmembers*

SUMMARY The scale of scientific data generated by experimental facilities and simulations in high-performance computing facilities has been proliferating with the emergence of IoT-based big data. In many cases, this data must be transmitted rapidly and reliably to remote facilities for storage, analysis, or sharing, for the Internet of Things (IoT) applications. Simultaneously, IoT data can be verified using a checksum after the data has been written to the disk at the destination to ensure its integrity. However, this end-to-end integrity verification inevitably creates overheads (extra disk I/O and more computation). Thus, the overall data transfer time increases. In this article, we evaluate strategies to maximize the overlap between data transfer and checksum computation for astronomical observation data. Specifically, we examine file-level and block-level (with various block sizes) pipelining to overlap data transfer and checksum computation. We analyze these pipelining approaches in the context of GridFTP, a widely used protocol for scientific data transfers. Theoretical analysis and experiments are conducted to evaluate our methods. The results show that block-level pipelining is effective in maximizing the overlap mentioned above, 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.
key words: high-performance data transfer, IoT-based big data, data integrity, pipelining

1. Introduction

With rapid advances in networking, computing, and electronic device technologies, the Internet has been applied to all significant parts of people's lives. Not only computers, but any device can also be connected to the Internet anytime and anywhere. For example, people can easily access any information and contact each other using smartphones or tablets. This has lead social network services (SNSs) to become very common (e.g., Facebook, Twitter).

Furthermore, the progress of the network and device technologies enable devices to be connected to the Internet; thereby, forming the Internet of Things (IoT) architectures [1]–[3]. Conventionally, this is referred to as Machine-to-Machine (M2M) services, which are served by

network-enabled electronic devices such as TVs, refrigerators, and air conditioners. Gradually, even tiny devices become part of the network. The acceleration of M2M services by ubiquitous computing has been extended to IoT. Thus, any object or device can be connected to the Internet; therefore, enabling data to be easily obtained or monitored in real-time. For example, the advent of IoT has enabled the effective monitoring of forest fires in real-time. Once Internet-connected fire/smoke sensors or cameras are installed in a forest, we can monitor the forest in real-time.

Interestingly, these Internet-related activities based on IoT services can generate a lot of real-time data; thus, causing a Big Data phenomenon. In general, the big data is defined by massive data sets that may be analyzed computationally to reveal patterns, trends, and associations, especially concerning human behavior and interactions [4]–[6]. Big data has four major characteristics, volume, velocity, variety, and veracity [5]–[7]. The volume refers to the size of generated data, the velocity relates to the speed of the generated data, the variety refers to the different forms/types of the generated data, and the veracity refers to the degree of inaccuracy in the generated data. That is, real-time IoT services can accumulate big data with very large, fast, various, and inaccuracy-prone aspects. Thus, it is necessary to determine the patterns/trends, to collect meaningful data sets, and to ensure the accuracy of the collected big data.

In the context of science, astronomical observation data is an example of real-time IoT-based big data [8], [9]. In general, the scale of scientific data generated by experimental facilities and simulations on high-performance computing environments has been growing rapidly. For example, the Dark Energy Survey (DES) telescope in Chile captures terabytes (TBs) of data each night. Another cosmology project, the Square Kilometer Array [10] will generate an exabyte every 13 days when it becomes operational in 2024. The Department of Energy (DOE) light source facilities generate tens of TBs of data per day now. This number is set to increase 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 large hadron collider (LHC) [11]. It is designed to record particle interactions occurring at its center. Every year, the CMS records and simulates six petabytes of proton-proton collision data to be processed and analyzed.

In terms of the veracity characteristic of the big data, it is essential to ensure the integrity of scientific data (e.g., astronomical observation) especially because these large

Manuscript received August 24, 2018.

Manuscript revised February 26, 2019.

Manuscript publicized April 24, 2019.

[†]The author is with Hongik University, South Korea.

^{††}The author is with Illinois Institute of Technology, USA.

^{†††}The author is with Argonne National Lab, USA.

^{††††}The author is with Changwon National University, South Korea.

*This article is the extended version of the conference paper: "Towards optimizing large-scale data transfers with end-to-end integrity verification," in Proc. 2016 IEEE International Conference on Big Data (Big Data), 2016.

a) E-mail: swchung@changwon.ac.kr (Corresponding author)

DOI: 10.1587/transinf.2018EDP7297

datasets are often transmitted over wide-area networks for multiple purposes, such as storage, analysis, and visualization. When transferring large quantities of data across end-to-end storage system-to-storage system paths, it is necessary to perform an end-to-end checksum verification. Even though some of the components in the end-to-end path implement their own data integrity check (these checks are insufficient). For example, the transfer control protocol (TCP) in network communication performs the TCP checksum [12], and storage controllers in data storage systems implement their own data integrity methods [13]. 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 as the number of components increase (a transfer involving 10 components, each with their integrity check that captures 99% of data corruption would result in 10% ($1 - 0.99^{10}$) of undetected data corruption).

J. Stone et al. [14] showed through extensive real-world experiments that the TCP checksum is not sufficient to guarantee end-to-end data integrity. A 16-bit checksum means that 1 in 65,536 bad packets will be erroneously accepted as valid. According to [15], approximately 1 in 5,000 Internet data packets is corrupted during transit. Thus, approximately 1 in every 300,000,000 ($65 \text{ K} \times 5 \text{ K}$) packets are accepted with corruption. It has been reported that an average of 40 errors per 1,000,000 transfers is detected on data transferred by the D0 experiment [16]. Projects such as DES require verification of checksums as part of their regular data movement process in order to detect file corruption due to software bugs or human error. To guarantee the data integrity despite network packet errors, we can take approaches of either integrity check at each of multiple data processing layers or the end-to-end integrity check. Due to redundant computation and still no guarantee on end-to-end integrity in case of integrity check at each of various data processing layers, we propose end-to-end integrity verification methods [17], [18].

While end-to-end data integrity check is crucial in big data transfer, it comes at a price. 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 [19], the analysis of GridFTP transfer logs indicates that the checksum overhead can be anywhere between 30% and 100%. In this article, we evaluate the 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 method, is not employed 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 scientific dataset, the improvement is up to 57% compared to the sequential execution and 47% compared to file-level pipelining.

Overall, our contribution in this article is three-fold. First, we empirically show that the end-to-end data integrity check in IoT-based big data transfer incurs considerable overhead while using the current file-level pipelining technique. Second, we propose a novel block-level pipelining method and compare it with the current file-level pipelining technique using to real experiments and mathematical analysis. Third, we improve the novel block-level pipelining technique by adaptively adjusting the pipeline stages based on whether the pipeline is checksum-dominant or transfer-dominant.

The rest of the article is organized as follows. In Sect. 2, we summarize the related work on high-performance data transfer and the associated data integrity issues. In Sect. 3, we describe pipelining approaches to optimize high-performance data transfer with an end-to-end data integrity check. In Sect. 4, we present the experimental results on real testbeds to evaluate the effectiveness of the pipelining approaches. We conclude with a summary of the work and future work in Sect. 5.

2. Related Work

Recently, Big Data has emerged as a hot topic, and many studies have discussed the definitions and basic concepts of big data [4]–[6]. In particular, the primary features of big data, namely, volume, velocity, variety, and veracity (4V), have also been discussed [5]–[7]. IoT-based big data is an important source of big data that has 4V features. An example of such IoT-based big data, astronomic observation data, is described in detail in [8], [9]. Often, data are transferred to remote sites for further in-depth analysis or data sharing for the research community. In general, the integrity of transferred data is not perfect, and a recent study reveals that 1 in 121 data transfers had at least one checksum error [20]. This article focuses on improving the performance of IoT-based big data transfers while guaranteeing data integrity.

Many tools have been developed for file transfers - GridFTP [21], Globus file transfer [19], bbcp [22], FDT [23], and XDD [24], to name a few. A number of approaches have been proposed to optimize large-scale wide-area data transfers. In [25], an algorithm that dynamically schedules a batch of data transfer requests to minimize the overall transfer time is proposed. The use of multiple TCP streams and concurrent file transfers are often required to achieve file transfer rates comparable to network speeds [26], [27]. Kettimuthu et al. incorporated on-the-fly checksum capabilities in GridFTP; however, it is not truly end-to-end in the sense that it does not account for any data corruption in the path between the host and the storage system at the data receiver. To address the need for end-to-end checksum as well as the limitations of the 16-bit TCP

checksum, Globus transfer service incorporated an additional 128-bit checksum computation (reduces the number of undetected bad packets to 1 in 2×10^{13}) by reading the file at the destination after it is written to the disk. Globus supports file-level pipelining for transfer and checksum by overlapping the checksum computation of a previously transferred file with the transfer of another file for multi-file transfers. Globus manages the data transfer logs for successful data transfers and integrity checks. We evaluate this file-level approach and perform comparisons to our novel block-level pipelining of transfer and checksum computation technique (which is not currently employed for production scientific data transfers, to the best of our knowledge). Table 1 summarizes the comparison of various data transfer tools with regard to data integrity check.

Many studies have been conducted on parallel data movement in distributed systems [28]–[32]. These studies usually address the issue of improving only data transfers by parallelizing and aggregating multiple data transfers. However, our work focuses more on the efficient end-to-end integrity verification method using parallel data integrity check and data transfer.

In [28], [33]–[36], IoT security issues were studied. However, it is not an easy and straightforward task, given that IoT architectures and devices have significant intrinsic variances and are heterogeneous with very limited computational capabilities. Nonetheless, several approaches have been introduced considering the characteristics of IoT. The major IoT security subjects (e.g., handling security keys, encrypting mechanisms, and cryptographic algorithms) have been presented in [28], [33]. In [34], [35], the authors discussed the layer-based IoT securities such as perception, network, and application layer. In [1], the application-specific securities are considered such that various IoT applications can be appropriately supported; however, primary security issues such as authentication, data integrity, and privacy are yet to be resolved. In [36], the specific network protocol facilitations for the IoT securities have been addressed. In this article, we focus on improving the efficiency of the integrity check for the end-to-end transferred IoT data. The data integrity check is relevant to guaranteeing secured data because both tasks, in general, utilize data encryption algorithms. Additionally, secured data encoding such as the secured hashing algorithm (SHA) [28] can also be combined

with our approach for secured data transfer.

3. 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. Figure 1 shows the comparison between network data integrity check and end-to-end data integrity check. The network data integrity check performs the data integrity only during network transfer. Even though other layers such as file systems can do an additional data integrity check, such layers still cannot guarantee the end-to-end data integrity [14]. Only the end-to-end data integrity check ensures that the stored files at the receiver are the same as the stored files at the sender. We propose high-performance data transfer methods combined with end-to-end data integrity check in this article. More specifically, we propose methods providing end-to-end data integrity check at the block level. We describe our proposed methods below in detail. First, we introduce the main “pipelining” strategy in our methods, which are classified into file-level pipelining and block-level pipelining. The analytical modeling for performance analysis is also established and illustrated in this part. Second, we explore the potential to enhance the block-level pipelining.

3.1 Pipelining

Pipelining is a useful parallelizing technique to improve the repetitive tasks composed of multiple steps. We apply pipelining to achieve high-performance data transfer with data integrity check. Figure 2 shows an example of the pipelining data transfer and data integrity check. T represents the data transfer, and C represents data the integrity check.

3.1.1 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, pipelined operations work best when all the operations take the same amount of time. In other words, the performance of the pipelined operations are dependent on operations (i.e.,

Table 1 Comparison of various data transfer tools with regard to data integrity check.

Level of Integrity Check	GridFTP	bbcp	FDT	XDD
Network data integrity check at the file level	Yes	Yes	Yes	No
End-to-end data integrity check at the file level	Yes	No	No	No
End-to-end data integrity check at the block level	No	No	No	No

Fig. 1 Network data integrity check vs. end-to-end data integrity check

data transfer and data integrity check) and the executions times vary depending on running 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 systems. 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 a 10 MB file transfer is overlapped with the data checksum for the previous file of size 10 GB (or vice versa), then the gap between transfer time and checksum time could be huge. This problem can be resolved in block-level pipelining, in which the gap (e.g., the difference in data transfer time for 10 MB and data checksum time for 10 MB if the block size is 10 MB.) is always constant.

3.1.2 Analytical Modeling

We analyzed 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 data checksum time (Transfer-Dominant Case), and 2) when data checksum time is longer than data transfer time (Checksum-Dominant Case). Based on tests, we found that both transfer time and checksum time (md5 sum) are a linear function of the data size in a relatively contention free environment.

Because it is hard to analytically model the performance of a dataset consisting of multiple random files, we generated two extreme cases of synthetic datasets having distinctive file size patterns. The first dataset consisted of twenty 10 GB files (20-10 GB). The second dataset consisted of ten repetitions of a 10 GB file and a 500 MB file (10 GB-500 MB), in which a 10 GB file transfer precedes

the transfer of a 500-MB file. We use these two datasets, to perform analytical modeling. We also experimentally evaluated the performance for these datasets and verified that the results agree with the analytical models. For mathematical analysis, we define t and c as follows.

- t : Transfer time for 500 MB of data
- c : Checksum time for 500 MB of data

For our experiments, we deliberately chose two different testbeds, one of which is transfer time dominant and the other is the checksum computation time dominant. Thus, we have separate analytical models for these two cases, as summarized in Table 2. The analytical formulations in Table 2 do not require the development of complex mathematical expressions. Consider the expression $400 \times t + 1/5 \times c$ for a 100 MB block-level pipeline in the 20-10 GB dataset in the transfer-dominant case. As the case is transfer-dominant, the checksum time is hidden by transfer time, which is the time taken for transferring a 100 MB block given by the following expression: $1/5 \times 500MB = 1/5 \times t$. A total transfer of 200 GB ($100MB \times 2000$) is performed by the transfer of 2000×100 MB blocks, followed by the last block's checksum computation, which takes $400 \times t + 1/5 \times c$ ($1/5 \times t \times 2000 + 1/5 \times c$). Note that the factor of $1/5$ comes from the fact that c is the checksum time for 500 MB and the checksum time for 100 MB is one-fifth of c because the checksum time is a linear function of the data size for md5sum. Note also that we transfer the whole file of 500 MB for the 10 GB-500 MB dataset for 100 MB block size while transferring 10 GB files in 100 MB blocks for simplicity. Each model is generated based on the number of pipeline stages in each case and the execution time for each stage. Both of the factors mentioned above are decided by the number of blocks and the transfer and checksum time of each block over 500 MB.

3.2 Improving Block-Level Pipelining

The analytical performance modeling of the block-level pipelining and general pipelining behaviors suggests that we can achieve the best performance when the data transfer time is approximately equal to the data checksum time. Hence, we explore possibilities to minimize the gap between the two operations as much as possible. In this study, the case in which the checksum and transfer times are similar is called the equilibrium case. Figure 3 shows that the transfer-dominant and checksum-dominant pipelines are transformed into the equilibril pipeline for better performance.

Fig. 2 Pipelining data transfer and data integrity check

Table 2 Comparison of analytical performance models of block-level pipeline, file-level pipeline, and sequential file transfer (existing method for baseline)

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

Fig. 3 Transforming a checksum-dominant pipeline into an equilibrated pipeline

3.2.1 Transforming a Checksum-Dominant Pipeline into an Equilibril Pipeline

In Checksum-Dominant Case, the time of the two operations can be approximately equal by reducing the data checksum time. We use multiple threads (and cores) to compute checksums of multiple parts of a block in parallel. However, the checksum computing time usually varies depending on the content of a block. To obtain predictable results, we must choose checksum computing algorithms whose computing time linearly increases with the block size. In this work, we reduce the gap between the block transfer time and block checksum time for Checksum-Dominant Cases by parallelizing the checksum computation.

3.2.2 Transforming a Transfer-Dominant Pipeline into an Equilibril Pipeline

In Transfer-Dominant Case, we reduce the transfer time by compressing the data size of a block. However, this approach also has issues because the varying compression ratio depends on the content of the block and the compression algorithm. In this work, we reduce the gap between the block transfer time and block checksum time for Transfer-Dominant Cases by parallelizing the compression computation.

4. Experimental Evaluation

In this section, we describe the testbeds for real-data transfer tests and present the experimental results followed by in-depth discussion. 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.

4.1 Experimental Testbeds

We evaluate the following three schemes:

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

We evaluated the block-level pipeline for various block sizes. For extensive experiments, we configured testbeds on both LAN and WAN environments. For a LAN environment, 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 selected two nodes (one sender and one receiver) on each cluster to run our tests. Because all nodes in a particular cluster have the same hardware configurations, we selected two nodes in the cluster for our experiments. For a WAN environment, we conducted partial experiments on the wide-area-network testbed comprised of two nodes – one is at the JLSE in the USA, and the other is at Hongik University in South Korea. The two nodes are connected via a 1-Gb research network connection, and the RTT between two nodes is approximately 160 ms. However, unlike the LAN environments, the network connection is not dedicated but shared by multiple users. Therefore, we averaged multiple runs to collect reliable data. This testbed is called the WAN testbed henceforth in this article, also called the Transfer-Dominant Case.

We used GridFTP toolkit for simulation. We installed GridFTP servers on both sender and receiver sites. Among GridFTP utilities, *globus-url-copy* is the command-line utility for data transfer, and *globus-url-copy* coordinated with two servers to move files from a sender site to a receiver site. We used *md5sum* for checksum computation because the algorithm shows that the computation time is a linear function of data size. We assumed that verification occurs once the checksum for data at both the sender and receiver sites is computed; further, the verification is merely the comparison of the checksum values. We also assumed that checksum errors do not occur during the simulation (It has been reported that an average of 40 errors per million transfers is detected on data transferred by the D0 experiment [19].)

In addition, we applied a simple simulation mechanism to implement multi-threaded checksum computation based on the Linux multi-thread scheduling policy. The basic Linux multi-thread scheduling policy tries to allocate different cores to multiple threads as long as the number of cores suffices. In our testbed, the cores of testbed machines

Table 3 Testbeds' hardware specifications

Testbed		Architecture	Processor	Memory/node	Network	File System/Storage
1) Cooley		Intel Haswell	Two 2.4 GHz Intel Haswell E5-2620 v3 processors per node (6 cores per CPU, 12 cores total)	384 GB RAM per node, 24 GB GPU RAM per node (12 GB per GPU)	10 Gbps	FDR Infiniband interconnect for GPFS
2) Rains		AMD Opteron	Four AMD Opteron 2216 processors per node (2 cores per CPU, 8 cores total)	8 GB	1 Gbps	20 Gb DDR InfiniBand interconnect for GPFS
3) WAN	Sender	Intel Xeon	Two Intel Xeon E5-2670 processors (8 cores per CPU, 16 cores total)	128 GB	1 Gbps	20 Gb DDR InfiniBand interconnect for GPFS
	Receiver	Intel Xeon	Two Intel Xeon E5-2640 processors (10 cores per CPU, 20 cores total)	128 GB	1 Gbps	SSD storage

are between 8 and 20. Therefore, the number of cores in our testbed is sufficient to simulate multi-threaded checksum computation by running multiple checksum threads on different partitions of a file.

We explain the testbed environments below, and we present the detailed specifications of each testbed in Table 3.

1. Cooley (Checksum-Dominant Case)

- Cooley [37] is an analysis and visualization cluster at ALCF.
- Two nodes in the Cooley testbed, a cluster of 126 nodes, is used.

2. Rains (Transfer-Dominant Case)

- Two nodes in the Rains testbed, a cluster of 16 nodes, are used.

3. WAN (Transfer-Dominant Case)

- One sender node at the JLSE in the USA and one receiver node at Hongik University in South Korea are used.

4.2 Evaluation Methodology

We employed GridFTP as a data transfer tool for our experiments. We have chosen GridFTP for two reasons. Firstly, some of us have been a part of GridFTP software development, which makes it easy to adjust parameters for experiments and modify the internal codes if needed. Secondly, GridFTP, on top of which Globus online web service is available, is the most successful data transfer tool in research communities compared to other tools in terms of subscribed users and business. This method can be applied to other data transfer tools without loss of generality. Globus transfer service [19] and *globus-url-copy* are the commonly used clients for GridFTP. Both support only file-level checksum; we used the latter for our tests. Because it supported only file-level checksum, we computed the checksums for all the three schemes by running the Linux system command *md5sum* in a separate thread. For multi-threaded checksum computation, we used external checksum computation emulation using Linux system command *md5sum* rather than modifying built-in checksum function in *globus-url-copy*. We verified that the performance of built-in checksum in

globus-url-copy is close to that of Linux system command *md5sum*.

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

1. 10 GB–500 MB dataset: Dataset with ten 10 GB files and ten 500 MB files, 105 GB in total.
2. 20–10 GB dataset: Dataset with twenty 10 GB files, 200 GB 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, 174 GB in total.

The first two datasets, 10 GB–500 MB and 20–10 GB, represent the extreme cases to demonstrate the effects of pipelining methods. The real dataset evaluates the performance of pipelining methods for real-world use cases. The 10–500 MB dataset is composed of files with only two sizes, one is large, and the other one is small. This type of dataset will benefit the most from the block-level pipeline. The 20–10 GB dataset is composed of files of the same size where the block-level pipeline benefits slightly over the file-level pipeline. The real dataset is composed of files with sizes ranging from megabytes to gigabytes. We expect the performance of the block-level pipeline for this case to be in between the block-level pipeline for 10–500 MB dataset and that for the 20–10 GB dataset.

We measured the performance of sequential file transfer, file-level pipeline transfer, and block-level pipeline transfer with block sizes of 100 MB, 500 MB, 1 GB, and 2 GB for all the experiments. We used block sizes of 50 MB, 100 MB, 500 MB, and 1 GB for the real dataset because the file size in the real dataset varied considerably from 10 MB to 2.1 GB with one-third of files less than 50 MB and only a few files larger than 2 GB.

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

Fig. 4 Performance comparison of sequential, file-level pipeline, and block-level pipeline (for different block sizes) on Cooley

methodology.

Suppose $T1$ is the time for transferring 20 blocks of the same size, e.g., 500 MB and $T2$ is the time for transferring one file with size 20 times the block size, (e.g., 10 GB = 500 MB x 20),

$$\text{Startup overhead} = \frac{T1 - T2}{19}$$

4.3 Block-Level Pipelining

4.3.1 Results on Cooley (Checksum-Dominant Case, $t < c$)

On the Cooley testbed, we measured $t \approx 0.5s$, $c \approx 1s$. We can obtain analytical performance as follows. Substituting t with value 0.5 and c with value 1 in the formulas in Table 2, we can calculate the approximate performance gain of the block-level pipelining over the file-level pipelining and file sequential transfers. For the 20–10 GB dataset, the maximum performance gain over the file-level pipelining is $\sim 10s$; For the 10 GB–500 MB dataset, the maximum performance gain over the file-level pipelining is $\sim 80s$. Compared to file sequential transfers, the block-level pipelining outperforms by $\sim 200s$ and $100s$, respectively, for 20–10 GB and 10 GB–500 MB datasets.

Figure 4 shows the performance of sequential transfer, 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 the 10–500 MB dataset, the performance gain of block-level pipeline over file-level transfers is $100s$ ($\sim 30\%$); For the 20–10 GB dataset, performance gain is less than $10s$; for the real dataset, the performance gain percentage ($\sim 16\%$) is in between the other two datasets. The 100-GB block size is the best for all three datasets. Moreover, performance seems to degrade with increasing block size with the exception of 1 GB performing better than 500 MB for the 20–10 GB dataset. We suspect that the transfer time is not perfectly linear with respect to the dataset size (the same is the case

Fig. 5 Performance comparison of sequential, file-level pipeline, and block-level pipeline (for different block sizes) on Rains

for checksum computation time) and the overlap between the transfer time and checksum computation time is best for the 100 MB block size.

4.3.2 Results on Rains (Transfer-Dominant Case, $t > c$)

On the Rains testbed, we measured $t \approx 7s$, $c \approx 1s$. We can obtain analytical performance as follows. Substituting t with value 7 and c with value 1 in the formulas in Table 2, we can calculate the approximate performance gain of block-level pipelining over file-level pipelining and file sequential transfers. For the 20–10 GB dataset, the maximum performance gain over file-level pipelining is $\sim 20s$; for the 10–500 MB dataset, the maximum performance gain over file-level pipelining is $\sim 200s$. Compared to file sequential transfers, the file-level pipelining outperforms by $\sim 400s$ and $200s$, respectively, for 20–10 GB and 10 GB–500 MB datasets.

Figure 5 shows the performance of sequential transfer, file-level pipeline, and block-level pipeline (for different block sizes) for different datasets on Rains. The results are consistent with the theoretical analysis: For the 10–500 MB dataset, the performance gain of block-level pipeline over file-level pipeline is $\sim 280s$ ($\sim 22\%$); for the 20–10 GB dataset, performance gain is less than $10s$; for the real dataset, the performance gain percentage ($\sim 14\%$) 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 Fig. 5, we observe that this is the case for 10–500 MB and 20–10 GB datasets, but for the real dataset, the performance of 1 GB block size is slightly worse than the others.

4.3.3 Equilibril Pipeline

The results of the previous experiments showed that the block-level pipeline transfer is an effective method to achieve high-performance transfer with data integrity check.

Fig. 6 Comparison of the performance of 1-Checksum-Thread and 2-Checksum-Thread on Cooley

However, we could not achieve an optimal pipeline owing to the difference between the block transfer time and block checksum computation time. In Checksum-Dominant Case, the checksum computation time is longer than the transfer time whereas, in Transfer-Dominant Case, the transfer time is longer than the checksum computation time. The unbalanced pipeline can be enhanced by reducing the longer processing element time. We conducted experiments to evaluate the effectiveness of balancing processing element times in a pipeline in both Checksum-Dominant Case and Transfer-Dominant Case.

(1) Transforming a Checksum-Dominant Pipeline into an Equilibril Pipeline

Next, we intend to observe the impact of the perfect pipeline. To achieve the perfect pipeline between block transfer and block checksum computation, we parallelized the checksum computation using two threads (cores). We have two checksum threads – one responsible for computing the checksum of the first half of a block and the other responsible for computing the checksum of the second half of the block. Figure 6 shows performance comparisons between the 1-Checksum-Thread Case and the 2-Checksum-Thread Case. The execution time of 2-Checksum-Thread is almost half of 1-Checksum-Thread in most block-level cases in each dataset except for the smallest block size (where the overhead of using two threads possibly dominates). In 2-Checksum-Thread Case, the 500 MB block size is the best for all three datasets (note that the 500 MB block size is the penultimate bar for the real dataset and the antepenultimate bar for the other two datasets). This is because the number of threads for checksum computation is selected in a manner to obtain a complete overlap for the 500 MB block size. Further, the 500 MB block size achieves almost a linear speedup with two threads. Other block sizes achieve significant performance improvement

with two threads for checksum computation as it dominates the transfer time in the Cooley testbed. The performance of the 2-Checksum-Thread Case is almost two times better than the 1-Checksum-Thread Case.

(2) Transforming a Transfer-Dominant Pipeline into an Equilibril Pipeline

Unlike the Cooley testbed, the Rains/WAN testbed is dominated by the transfer time. Based on the results on Cooley, we believed that similar performance gain could be achieved if the transfer time can be reduced to match the checksum computation time. Compressing the data prior to transferring over the networks is one possible way of reducing the transfer time when the network bandwidth is a bottleneck. We evaluated this approach on the WAN testbed where two nodes are connected via a shared 1 Gb network path. Because the RTT on the WAN testbed is ~160 ms, the startup time due to TCP slow start takes longer than in LAN environments, which also worsens data transfer throughput on the WAN testbed. We evaluated only with the 20–10 GB dataset for proof-of-concept. The experimental results are shown in Fig. 7. The first row in the x-axis denotes block sizes for block pipeline and file size for file-based sequential/pipeline transfer. The second row in the x-axis denotes whether the transfer is block pipeline transfer or file-based transfer. The last row in the x-axis denotes whether the transfer is pipeline transfer or sequential transfer. The blue bars represent the normal transfer, and the orange bars represent the compressed transfer. For example, of the last two bars, the blue bar represents the transfer time of sequential file-based normal transfer, and the orange bar represents the transfer time of sequential file-based compressed transfer.

The results in Fig. 7 show that reducing the transfer time by compressing data on the WAN testbed improves the overall transfer time. However, performance improvement is dependent on the compressibility of the dataset and

Fig. 7 Performance comparison of normal transfer and compressed transfer in sequential, file-level pipeline, and block-level pipeline (for different block sizes) on the WAN testbed

available CPU resources. In our experiments, because the data are readily compressible and sufficient CPU resources are available for compression, we could improve the overall transfer time. Compared with the baseline test (file-based sequential transfer), the block pipeline transfer with compression could reduce almost 30% transfer time. In general, data transfer with data integrity verification is involved in various computer resources. The number of CPU cores is related to possible improvement through data compression. The network speed may be the bottleneck for the overall data transfer if all the other resources are sufficiently provided, and the minimum size of TCP buffer memory should be reserved to guarantee high network bandwidth utilization. In terms of equilibrated pipeline, the checksum-dominant pipeline and transfer-dominant pipeline can be improved by the increased number of cores on which multiple checksum/compression threads can run concurrently.

4.4 Discussion

What if a block goes through multiple pipeline steps such as a block generation step? Even though the block generation time is not considered in this article, there is a way to apply the proposed method to such a case. For example, if a block goes through three steps (generation, transfer, and data integrity check), we can merge the first two steps and regard two steps as one transfer step. In case that we really want to deal with three steps separately, we can also take a similar approach as transforming a three-step pipeline into an equilibrated pipeline.

Regarding transfer-dominant pipeline, as the compression time grows, it becomes hard to transform a transfer-dominant pipeline into an equilibrated pipeline. For such reasons, the experiments were conducted on the WAN environment where the transfer time is much longer than the LAN environment, and it is highly likely that the compression will help transform into an equilibrated pipeline. However, assuming that compression time will linearly decrease as the

more number of cores/threads are used (this is the case of *md5sum*), we can expect the negative impact of compression time to be nullified to some extent.

The block-level pipeline benefits transferring datasets composed of various file sizes. As analyzed in Table 2, in both cases of the transfer-dominant pipeline and checksum-dominant pipeline, the block-level pipeline transfer outperforms the file-level pipeline transfer, especially for the 10GB-500MB dataset. The 10GB-500MB dataset is composed of alternating a 10 GB file and a 500 MB file whereas the 20-10GB dataset is just twenty files of same size 10 GB. Intuitively, this makes sense because alternating different sized files make it difficult to make an equilibrated pipeline. In experiments, Figs. 4 and 5 show the same results as the analytical results, i.e., in case of the 10G-500M dataset, the block-level pipeline outperforms the file sequential transfer and the file pipeline transfer. Therefore, we can infer that regarding datasets composing of various sized files, we should prefer the block-level pipeline transfer to the file-level pipeline transfer, and we should select the block size such that the multiples of the block size should be sizes of most files.

5. Conclusion

To effectively support the veracity feature of the IoT-based big data, we proposed an efficient scheme to promote end-to-end integrity verification using climate data. Specifically, the work presented herein is a summary of our work on block-level pipelining to overlap data transfer and checksum computation. Based on the theoretical analysis and experimental results, we concluded that the block-level pipeline is an effective approach to optimize data transfers with end-to-end integrity checking. We further showed that block-level pipeline could 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-scientific dataset. The results obtained the motivation to explore more optimized methods based on current work. The experimental results demonstrated that the performance of block-level pipelining varies for different block sizes. In addition, as per the 2-Checksum-Thread experiment results, highest performance gains can be achieved when the transfer time and checksum time match (or are approximately equal). We intend to study how to determine the appropriate block size, data integrity algorithm (in addition to MD5, other data integrity algorithms such as CRC [38], adler32 [39] are used for wide-area data transfers), data compression methods, and 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, considering various IoT-based big data applications.

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. This work was also supported by the Hongik University new faculty research support fund.

References

- [1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol.54, no.15, pp.2787–2805, 2010.
- [2] C.-W. Tsai, C.-F. Lai, and A.V. Vasilakos, "Future Internet of Things: open issues and challenges," *Wireless Networks*, vol.20, no.8, pp.2201–2217, Nov. 2014.
- [3] M. Kim, H. Ahn, and K.P. Kim, "Process-Aware Internet of Things: A Conceptual Extension of the Internet of Things Framework and Architecture," *KSII Transactions on Internet and Information Systems*, vol.10, no.8, pp.4008–4022, Aug. 2016.
- [4] J.S. Ward and A. Barker, "Undefined by data: A survey of big data definitions," arXiv:1309.5821 [cs], Sept. 2013. arXiv: 1309.5821.
- [5] H. Hu, Y. Wen, T.-S. Chua, and X. Li, "Toward Scalable Systems for Big Data Analytics: A Technology Tutorial," *IEEE Access*, vol.2, pp.652–687, 2014.
- [6] S.R. Jeong and I. Ghani, "Semantic Computing for Big Data: Approaches, Tools, and Emerging Directions (2011–2014)," *KSII Transactions on Internet and Information Systems*, vol.8, no.6, pp.2022–2042, June 2014.
- [7] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, and B. Qiu, "BigDataBench: A big data benchmark suite from internet services," 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), pp.488–499, Feb. 2014.
- [8] A. Jacobs, "The Pathologies of Big Data," *Commun. ACM*, vol.52, no.8, pp.36–44, Aug. 2009.
- [9] E.D. Feigelson and G.J. Babu, "Big data in astronomy," *Significance*, vol.9, no.4, pp.22–25, Aug. 2012.
- [10] R. Spencer, "The square kilometre array: The ultimate challenge for processing big data," IET Seminar on Data Analytics 2013: Deriving Intelligence and Value from Big Data, pp.1–26, Dec. 2013.
- [11] N. Magini, N. Ratnikova, P. Rossman, A. Sánchez-Hernández, and T. Wildish, "Distributed data transfers in CMS," *Journal of Physics: Conference Series*, vol.331, no.4, p.042036, Dec. 2011.
- [12] T.C. Maxino and P.J. 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.
- [13] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, "The new ext4 filesystem: current status and future plans," *Proc. Linux Symposium*, pp.21–33, Citeseer, 2007.
- [14] J. Stone and C. Partridge, "When the CRC and TCP checksum disagree," *ACM SIGCOMM Computer Communication Review*, pp.309–319, ACM, 2000.
- [15] V. Paxson, "End-to-end internet packet dynamics," *Proc. ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '97, New York, NY, USA, pp.139–152, ACM, 1997.
- [16] A. Baranovski, K. Beattie, S. Bharathi, J. Boverhof, J. Bresnahan, A. Chervenak, I. Foster, T. Freeman, D. Gunter, K. Keahey, C. Kesselman, R. Kettimuthu, N. Leroy, M. Link, M. Livny, R. Madduri, G. Oleynik, L. Pearlman, R. Schuler, and B. Tierney, "Enabling petascale science: data management, troubleshooting, and scalable science services," *Journal of Physics: Conference Series*, vol.125, p.012068, July 2008.
- [17] J.H. Saltzer, D.P. Reed, and D.D. Clark, "End-to-end Arguments in System Design," *ACM Trans. Comput. Syst.*, vol.2, no.4, pp.277–288, Nov. 1984.
- [18] T. Moors, "A critical review of "End-to-end arguments in system design"," 2002 IEEE International Conference on Communications. Conference Proceedings. ICC 2002 (Cat. No.02CH37333), vol.2, pp.1214–1219, April 2002.
- [19] 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.
- [20] Z. Liu, R. Kettimuthu, I. Foster, and N.S.V. Rao, "Cross-geography Scientific Data Transferring Trends and Behavior," *Proc. 27th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '18, New York, NY, USA, pp.267–278, ACM, 2018. event-place: Tempe, Arizona.
- [21] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The globus striped GridFTP framework and server," *Proc. 2005 ACM/IEEE conference on Supercomputing*, SC '05, Washington, DC, USA, pp.54–64, 2005.
- [22] "bbcp," <http://www.slac.stanford.edu/~abh/bbcp/>.
- [23] "Fast Data Transfer," <http://monalisa.cern.ch/FDT/>.
- [24] 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," *IEEE/NASA Goddard Conference on Mass Storage Systems and Technologies*, vol.0, pp.1–6, 2011.
- [25] 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," *IEEE International Symposium on Parallel and Distributed Processing*, 2008. IPDPS 2008. pp.1–12, IEEE, 2008.
- [26] 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.
- [27] 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," *TERENA Networking Conference (TNC)*, 2013.
- [28] H. Suo, J. Wan, C. Zou, and J. Liu, "Security in the Internet of Things: A Review," 2012 International Conference on Computer Science and Electronics Engineering, pp.648–651, March 2012.
- [29] H. Bui, E.-S. Jung, V. Vishwanath, A. Johnson, J. Leigh, and M.E. Papka, "Improving sparse data movement performance using multiple paths on the Blue Gene/Q supercomputer," *Parallel Computing*, vol.51, pp.3–16, Jan. 2016.
- [30] F. Tessier, P. Malakar, V. Vishwanath, E. Jeannot, and F. Isaila, "Topology-aware Data Aggregation for Intensive I/O on Large-scale Supercomputers," *Proc. First Workshop on Optimization of Communication in HPC, COM-HPC '16*, Piscataway, NJ, USA, pp.73–81, IEEE Press, 2016.
- [31] F. Tessier, V. Vishwanath, and E. Jeannot, "TAPIOCA: An I/O Library for Optimized Topology-Aware Data Aggregation on Large-Scale Supercomputers," 2017 IEEE International Conference on Cluster Computing (CLUSTER), pp.70–80, Sept. 2017.
- [32] Y. Kim, S. Atchley, G.R. Vallée, S. Lee, and G.M. Shipman, "Optimizing End-to-End Big Data Transfers over Terabits Network Infrastructure," *IEEE Trans. Parallel Distrib. Syst.*, vol.28, no.1, pp.188–201, Jan. 2017.
- [33] S. Sicari, A. Rizzardi, L.A. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: The road ahead," *Computer Networks*, vol.76, pp.146–164, Jan. 2015.
- [34] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan, "Internet of things (IoT) security: Current status, challenges and prospective measures," 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST), pp.336–341, Dec. 2015.
- [35] K. Zhao and L. Ge, "A Survey on the Internet of Things Security," 2013 Ninth International Conference on Computational Intelligence

and Security, pp.663–667, Dec. 2013.

- [36] J. Granjal, E. Monteiro, and J.S. Silva, “Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues,” *IEEE Communications Surveys Tutorials*, vol.17, no.3, pp.1294–1312, 2015.
- [37] “Cooley,” <http://www.alcf.anl.gov/user-guides/cooley>.
- [38] J.S. Sobolewski, “Cyclic redundancy check,” *Encyclopedia of Computer Science*, pp.476–479, John Wiley and Sons, Chichester, UK, 2003.
- [39] P. Deutsch and J.L. Gailly, “Zlib compressed data format specification version 3.3,” Tech. Rep., RFC Editor, 1996.

Eun-Sung Jung is now an assistant professor in the Department of Software and Communications Engineering at Hongik University. He earned a Ph. D. in the Department of Computer and Information Science and Engineering at the University of Florida. He also received B.S. and M.S. degrees in electrical engineering from Seoul National University, Korea, in 1996 and 1998, respectively. He was a postdoctoral researcher in the Mathematics and Computer Science Division at Argonne National Laboratory

from 2013 to 2016. He also held a position of a research staff member at Samsung Advanced Institute of Technology from 2011 to 2012. His current research interests include high-performance data processing/transfer, IoT streaming data analytics, cloud computing, network resource/flow optimization. He has published over 30 research papers in conference proceedings and journals.

Si Liu is a software engineer with HERE Technologies in Chicago, USA. She earned M.S. degrees in computer science from Illinois Institute of Technology, USA, in 2017. She now works in the Highly Automated Driving (HAD) Content Pipeline team as a big data software Engineer.

Rajkumar Kettimuthu is a project leader in the Mathematics and Computer Science Division at Argonne National Laboratory and a fellow at University of Chicago’s Computation Institute. His research interests include transport protocols for high-speed networks; research data management in distributed systems; and the application of distributed computing to problems in science and engineering. He is the technology coordinator for Globus GridFTP, a widely used data movement tool. He has published over 60 articles in parallel, distributed and high performance computing. He is a senior member of IEEE and ACM.

Sungwook Chung is an associate professor in the Department of Computer Engineering at Changwon National University, S. Korea. He received M.S. and Ph.D. degrees in the Computer and Information Science and Engineering (CISE) department from the University of Florida, USA, in 2005 and 2010, respectively. He worked for smart IPTV developments at the KT Central R&D Lab. in S. Korea from 2010 to 2012. His research interests include IoT-based distributed multimedia systems and community

area networks for high-quality realtime content sharing and streaming services.