Original Article

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
1–16

DOI: 10.1177/1094342015618017
hpc.sagepub.com
SAGE

Enhance parallel input/output with cross-bundle aggregation

Teng Wang[1], Kevin Vasko[2], Zhuo Liu[2], Hui Chen[2] and Weikuan Yu[1]

Abstract

The exponential growth of computing power on leadership scale computing platforms imposes grand challenge to scientific applications' input/output (I/O) performance. To bridge the performance gap between computation and I/O, various parallel I/O libraries have been developed and adopted by computer scientists. These libraries enhance the I/O parallelism by allowing multiple processes to concurrently access the shared data set. Meanwhile, they are integrated with a set of I/O optimization strategies such as data sieving and two-phase I/O to better exploit the supplied bandwidth of the underlying parallel file system. Most of these techniques are optimized for the access on a single bundle of variables generated by the scientific applications during the I/O phase, which is stored in the form of file. Few of these techniques focus on cross-bundle I/O optimizations. In this article, we investigate the potential benefit from cross-bundle I/O aggregation. Based on the analysis of the I/O patterns of a mission-critical scientific application named the Goddard Earth Observing System, version 5 (GEOS-5), we propose a Bundle-based PARallel Aggregation (BPAR) framework with three partitioning schemes to improve its I/O performance as well as the I/O performance of a broad range of other scientific applications. Our experiment result reveals that BPAR can deliver 2.1 × I/O performance improvement over the baseline GEOS-5, and it is very promising in accelerating scientific applications' I/O performance on various computing platforms.

Keywords

Aggregation, GEOS-5, high-performance computing, parallel I/O, scientific application, storage system

I. Introduction

The computing power of the leadership scale high-performance computing (HPC) systems have maintained the exponential growth over the past 10 years. For instance, the computing power of the Cray supercomputer increased from 318Tflops/s in 2006 (Cray-XT4) to 1.3Pflops/s in 2008 (Cray-XT5), then jumped all the way to 20Pflops/s (Cray-XK7). This degree of computing power allows scientific applications to solve complex problems at massive scale during the computation phase, such as weather forecasting, simulation of nuclear fusion, and so on. Each round of computation will produce colossal volume of scientific variables and these variables are usually shared by all the processes. After the computation phase, these variables are packed into bundles, all the bundles are then written to the parallel file system (PFS) during the ensuing input/output (I/O) phase, each stored as a file. The ultrahigh computing power and gigantic volume of data set generated pose a daunting challenge to the computer scientists in how to garner commensurate I/O bandwidth to store these bundles during the I/O phase.

Many efforts, both past and present, have been invested to bridge the performance gap between computation and I/O. Consequently, various parallel I/O libraries are developed and adopted, such as the MPI-IO (Corbett et al., 1996), Parallel NetCDF (PnetCDF) (Li et al., 2003), and HDF5 (HDF5 Home Page, n.d.). These libraries enhance the I/O parallelism by allowing multiple processes to concurrently access the shared data set. Meanwhile, they are integrated with a set of I/O optimizations such as data sieving (Thakur et al., 1999), two-phase I/O (Thakur et al., 1999), and chunking (Sarawagi and Stonebraker, 1994) to better exploit the supplied bandwidth of the underlying PFS.

[1]Florida State University, Tallahassee, FL, USA
[2]Auburn University, Auburn, AL, USA

Corresponding author:
Weikuan Yu, Florida State University, 1017 Academic Way, Tallahassee, FL 32306.
Email: yuw@cs.fsu.edu

Most of these techniques are optimized for the access of a single shared bundle. This is achieved by associating the interface with the file descriptor of this bundle. For instance, MPI-IO defines the collective I/O operations that allow all the processes to collaboratively transform small, noncontiguous I/O requests associated with the same shared bundle file to large, contiguous I/O requests. However, few of these I/O techniques consider cross-bundle optimizations that can lead to I/O performance improvements for scientific applications as they generally access more than one bundle during their life cycle. For instance, the life cycle of the Goddard Earth Observing System, version 5 (GEOS-5) application (Liu et al., 2013b) is composed of several time steps. In each time step, it will generate multiple bundle files. Cross-bundle optimization allows processes to collaboratively work on all the bundles concurrently, which extends the optimization scope and allows for higher potential I/O performance improvement.

This article aims to explore the potential benefit from cross-bundle I/O aggregation. Our study is based on a mission-critical application named GEOS-5. The I/O technique of baseline GEOS-5 is parallel I/O using multiple NetCDF files. Namely, each bundle file is assigned a different master process. All the processes first send their bundle data to the master, the master then writes the bundle data to storage. The use of multiple files allows writing on each bundle file to be conducted concurrently. However, the serial NetCDF employed by the baseline GEOS-5 only allows one process to write on each file, yielding limited parallelism. In addition, since each master process needs to receive its bundle data from all other processes, the writing on each bundle cannot be fully parallelized, and such all-to-one communication can result in heavy contention.

Our early attempt replaced serial NetCDF with PnetCDF. PnetCDF allows each process to concurrently operate on the same file, thereby improving the parallelism. In our experiment, we observed that GEOS-5 with PnetCDF initially delivered promising performance; however, it did not scale well due to the heavy contention and metadata overhead with a large number of processes (Li et al., 2003). In addition, the original data format of GEOS-5 is not maintained by PnetCDF.

Therefore, a new solution that preserves the baseline data format of GEOS-5 is needed with enhanced parallelism and reduced contention at scale. In this article, we propose a Bundle-based PARallel Aggregation (BPAR) framework with three of its partitioning schemes that can be applied to a variety of scientific applications. BPAR associates each file with a distinct group of processes that can concurrently work on each file, thereby improving the parallelism with decoupled I/O on each file. Meanwhile, the smaller group size avoids the heavy communication and I/O contention introduced by the participation of all the processes. For the experiment, we have implemented BPAR on top of GEOS-5. The results of our experiment reveal that GEOS-5 with BPAR can achieve on average 2.1× I/O performance improvement over the baseline GEOS-5.

The advantage of BPAR on GEOS-5 also raises three questions: First, GEOS-5 is designed to run on the Discover supercomputer, which assumes General PFS (GPFS) as the underlying PFS. Is BPAR also beneficial for applications running on other HPC system with a different file system? Second, GEOS-5 output fixed number of bundle files (7) during each I/O phase. How is the performance of BPAR affected by the varying number of shared files output at each I/O phase? Third, BPAR demonstrates significant improvement over the I/O techniques harnessed by GEOS-5. How is BPAR compared with other widely used parallel I/O techniques such as MPI-IO? To answer these questions, we have also characterized the performance of BPAR on Discover and Titan in writing different number of shared files and compared its performance with ROMIO (Thakur et al., 1999), a standard MPI-IO implementation that serves as the underlying library for many other parallel I/O techniques. Our result demonstrates BPAR achieves on average 17.6% I/O performance improvement over ROMIO on Discover and yields on average 5× the bandwidth of ROMIO on Titan. Taken together, we have made the following contributions.

- We have systematically analyzed the I/O performance of the baseline GEOS-5 and GEOS-5 with PnetCDF and identified the major factors that restrict their performance.
- Based on our analysis, we have proposed BPAR with three of its partitioning schemes to accelerate the I/O performance of GEOS-5. Our experiment results reveal BPAR is able to achieve 2.1× I/O performance improvement over the baseline GEOS-5.
- To evaluate BPAR on the different HPC systems and observe its potential benefit for other applications, we have characterized its performance on Discover and Titan and compared its performance with ROMIO. Our experiment results reveal BPAR efficiently outperforms ROMIO on these systems.

The rest of this article is organized as follows. Section II analyzes the I/O performance of baseline GEOS-5 and GEOS-5 with PnetCDF. We then introduce BPAR framework in Section III, followed by Section IV that proposes the three partitioning schemes under BPAR. Section V evaluates the performance of BPAR on GEOS-5 and characterizes its performance on different HPC systems. Section VI reviews the

related work. Finally, we conclude the article with some future work in Section VII.

II Background and motivation

In this section, we provide a brief overview of GEOS-5 and its data organization. We then present and analyze the I/O performance of the baseline GEOS-5 and GEOS-5 with PnetCDF.

A Overview of GEOS-5

GEOS-5 is being developed by National Aeronautics and Space Administration (NASA) to support the earth science research. It simulates climate changes that span diverse temporal granularities, from hours to multiple centuries.

The simulation data set is organized with NetCDF-4 format. The entire data space is divided into what GEOS-5 calls collections, also referred to as data bundles. Each bundle describes certain climate systems, such as moisture and turbulence. It consists of a mixture of multiple variables. These variables are multidimensional data sets, either formatted as 3-D variables transposing into latitude, longitude, and elevation or 2-D variables represented by latitude and longitude. These variables define disparate aspects of the model, such as cloud condensates and precipitation.

GEOS-5 applies a 2-D domain decomposition to all variables among parallel processes. Each 2-D variable contains one 2-D plane. A 3-D variable consists of multiple 2-D planes. Every plane is evenly distributed to all the processes. Such data organization is simplified in Figure 1. PX refers to process X. Bundle1 and Bundle2 are the two bundles written in the I/O phase, Bundle1 contains two 2-D variables (var1 and var3), each holding one plane. Var2 is a 3-D variable composed of two planes. Similarly, Bundle2 also incorporates both the 2-D (var1 and var2) and 3-D variables (var3).

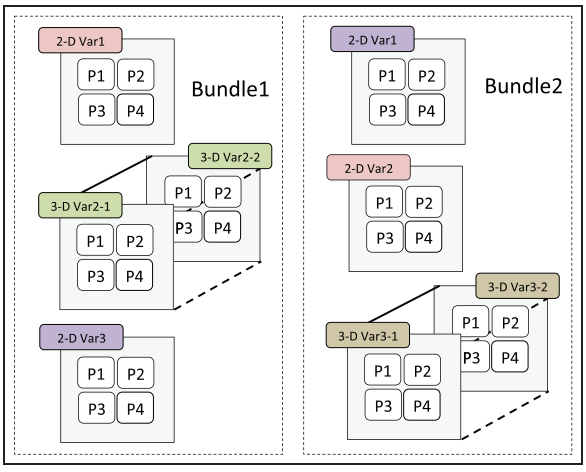

Figure 1. The data organization of GEOS-5. GEOS-5: Goddard Earth Observing System, version 5.

The life cycle of GEOS-5 alternates between computation phase and I/O phase. During each I/O phase, the state variables are written to the underlying PFS for post-processing and restart after failure. To maintain data integrity, all state variables within the same bundle are written to a shared bundle file.

B Baseline NetCDF-based GEOS-5 I/O

In the baseline GEOS-5 implementation, a master process is elected for each bundle to handle all the I/O requests from other processes for the bundle. Each bundle is written plane by plane using serial NetCDF. Namely, after the master process receives each plane data from the other processes, it writes the plane to the file system and proceeds to handle the next plane. So the I/O time is dominated by the communication time (time spent for receiving plane data) and write time.

There are two major drawbacks of such technique. First, it cannot scale well. This is because writing each bundle involves the participation of all the processes. The growing number of processes can soon overwhelm the masters' limited resource with the surging number of concurrent requests. Second, it does not yield good parallelism. Since all processes are involved in writing each bundle, every process cannot proceed to the next bundle until it completes its work on the previous bundle.

C GEOS-5 with PnetCDF

GEOS-5 with PnetCDF replaces serial NetCDF with PnetCDF for enhanced parallelism. Like other parallel I/O techniques, PnetCDF allows all the processes to concurrently write their share of the entire data set to the file system. In GEOS-5, each process possesses one piece of the plane data, and PnetCDF allows the processes to directly write their piece without explicit communication with other processes. One drawback of PnetCDF is that its data format does not match the data format of NetCDF if NetCDF uses HDF5 as the underlying library, like in GEOS-5. Besides, there are some other potential factors that affect its performance, such as the high metadata overhead in header I/O and heavy I/O contention with a large number of processes.

D Analyzing the I/O performance of baseline NetCDF-based GEOS-5 and GEOS-5 with PnetCDF

To analyze the I/O performance of baseline NetCDF-based GEOS-5 and GEOS-5 with PnetCDF, we have systematically benchmarked their performance. As shown in Figure 2, these two implementations are

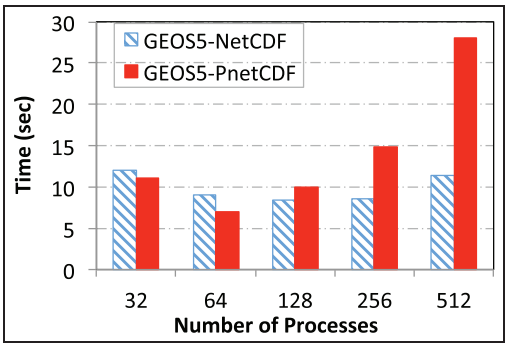

Figure 2. The I/O performance of baseline NetCDF-based GEOS-5 and GEOS-5 with PnetCDF. I/O: input/output.

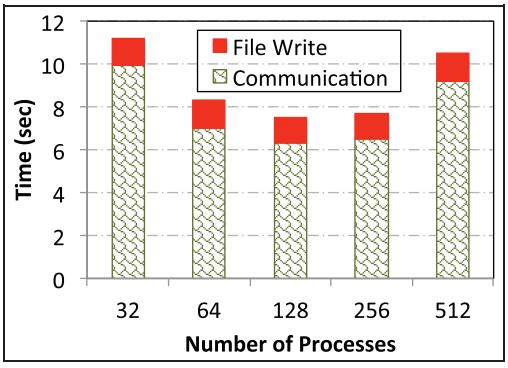

Figure 3. Dissection of baseline NetCDF-based GEOS-5 I/O. GEOS-5: Goddard Earth Observing System, version 5.

denoted respectively by GEOS-5-NetCDF and GEOS-5-PnetCDF.

As shown in Figure 2, GEOS-5 with PnetCDF initially delivers promising performance, but its I/O time is prolonged, as more processes are involved. On the other hand, the time of baseline NetCDF-based GEOS-5 I/O firstly decreases from 32 to 128 processes then keeps on increasing for executions with processes more than 128.

We further dissect the I/O time for a more elaborate analysis. Figure 3 reveals the detailed time dissection of baseline NetCDF-based GEOS-5 I/O, and the communication times (defined in Section II-B) and write times of the last master process that completed I/O are also presented in Figure 3. It can be observed that the I/O time is dominated by the communication time. It first decreases and then is prolonged as the number of processes increases. While the decreased communication time is due to the bandwidth of master process not being fully utilized with fewer number of processes, the reverse trend with more process count results from exacerbated contention on the master's limited resource. In contrast, the write time is negligible due to the high aggregated I/O throughput rendered by GPFS on Discover (Discover Supercomputer Statistics, n.d.).

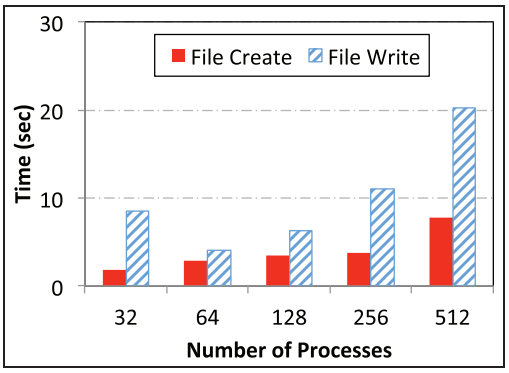

Figure 4. Dissection of GEOS-5 with PnetCDF. GEOS-5: Goddard Earth Observing System, version 5.

The dissection of GEOS-5 with PnetCDF is shown in Figure 4. It can be observed the two primary I/O operations (file write and file create) both impose non-negligible overhead. When the number of processes increases, the file write time firstly drops then grows sharply. The decreased write time from 32 to 64 processes is the result of more aggregated throughput delivered by the doubled number of writers. The increased write time from 64 to 512 processes is the result of the exacerbated contention from more and more processes. In particular, GEOS-5 writes multiple bundle files in each time step. Since each plane is shared by all the processes, when there is a large number of processes, the small I/O requests from each process lead to heavy I/O contention. Such I/O contention persists through all these bundle files. Also, PnetCDF guarantees strong data consistency in header I/O. The frequent and heavy metadata synchronization for maintaining such consistency serves as the major reasons for a long file creation time.

As a summary, both baseline NetCDF-based GEOS-5 and GEOS-5 with PnetCDF have multiple drawbacks. Baseline NetCDF-based GEOS-5 I/O has the constraint of limited scalability and parallelism. Although GEOS-5 with PnetCDF benefits from the increased parallelism, its performance is constrained due to the heavy I/O contention in each collaborative operation at scale. In addition, the requirement for strong consistency can incur nonnegligible overhead to PnetCDF.

III Our proposed solution: A high-level overview of BPAR

From the above discussion, we identified the major factors that constrain the I/O performance of baseline NetCDF-based GEOS-5 and GEOS-5 with PnetCDF. Therefore, we propose a Bundle-based PARallel aggregation framework called BPAR that mitigates the issues of these two techniques.

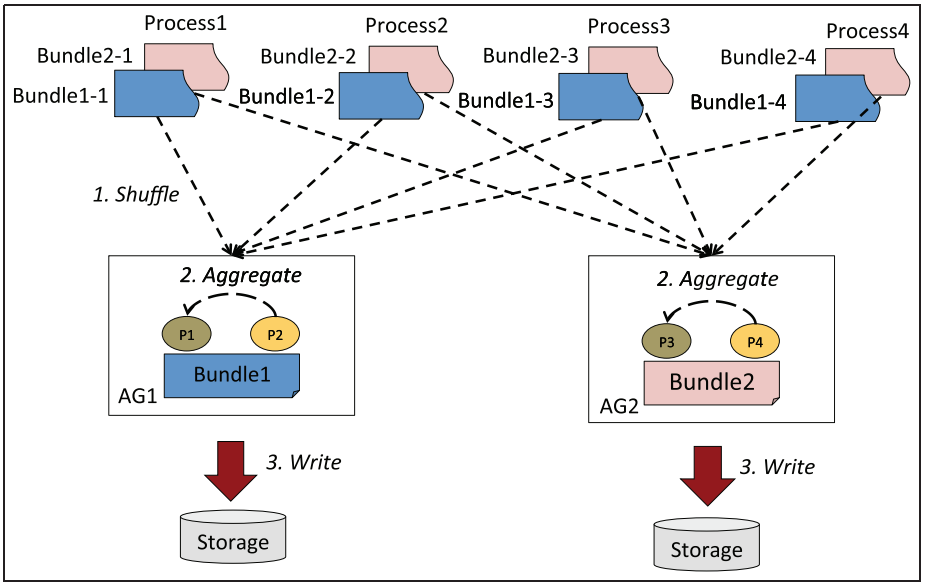

Figure 5. A high-level overview of BPAR. BPAR: Bundle-based PARallel Aggregation.

The main idea of BPAR is to parallelize the I/O of different bundle files by assigning each bundle with a distinct set of processes in order that each set of processes can perform I/O for its bundle independently and concurrently. In this way, the smaller number of processes in each group mitigates the communication and I/O contention in both baseline NetCDF-based GEOS-5 I/O and PnetCDF.

Figure 5 demonstrates the high-level overview of BPAR through an example. In this example, two bundles are to be processed by four processes, and each bundle's data are initially distributed among all the four processes. According to a certain partitioning technique, the four processes are divided into two aggregation groups (AGs), each of which takes charge of the I/O operation for one bundle. After all the processes complete a round of data shuffling operation, each bundle is entirely possessed by its designated AG, for example, bundle1's data possessed by AG1 composed of processes P1 and P2. Then, one aggregator process is elected in each AG (processes P1 and P3 in Figure 5) to further aggregate the data inside each AG and then write the collected data to the storage. There can be multiple aggregators depending on the workload. Due to the limited memory of aggregators, the aggregation and write operation can interleave with each other.

BPAR can be applied to a wide range of scientific applications that output several bundle files across I/O phases or inside each phase. On one hand, many scientific applications (e.g. S3D, Sankaran et al., 2008 and NAS Parallel Benchmarks, Wong and der Wijngaart, 2003) alternate between computation phases and I/O phases. During I/O phase, in-memory variables of all processes are written to file systems as one shared bundle file (N-1 write) or separate bundle files for each process (N-N write). A good way to accelerate I/O for N-1 case is to buffer the shared files of multiple phases and then output these files to PFS using BPAR. On the other hand, scientific applications such as GEOS-5 can choose to generate multiple shared bundle files during each I/O phase, each bundle containing a set of correlated variables. BPAR also fits these scientific applications to accelerate I/O in a single I/O phase.

A Major procedures that affect I/O performance

From the above discussion, the I/O time for the bundle *i*—*T*i is determined by its data shuffle time, aggregation time, and write time, denoted as *T*sfl, *T*agg, and *T*w, respectively. Thus we can approximately calculate *T*i using the following equation:

$$T_{\mathrm{i}} = T_{\mathrm{sfl}} + T_{\mathrm{agg}} + T_{\mathrm{w}} \tag{1}$$

Therefore, the total I/O time for writing all B bundles *T*io is represented using the following equation:

$$T_{\mathrm{io}} = \max(T_1, T_2, \ldots, T_{\mathrm{B}}) \tag{2}$$

The performance of three procedures are directly determined by the group partitioning strategy, that is, how many processes and which processes are to be placed in each bundle's aggregation group. For instance, placing the processes that reside on the same physical node inside the same AG may improve *T*agg due to enhanced locality; however, it may overprovision some small bundles with extra processes and prolong the I/O time of large bundles.

B Relationship of BPAR and MapReduce

MapReduce (Dean and Ghemawat, 2008), as one of the representative parallel programming models, has received widespread success on the cloud platforms. These platforms collocate compute and storage resources on the same node, a paradigm referred to as data-centric paradigm (Wang et al., 2014a). However, it has received limited applications on HPC systems that follow the compute-centric paradigm, which separates the computing and storage resources in the form of computing clusters and PFSs. Although BPAR mainly serves scientific applications on the HPC systems, its functionality can be accomplished by existing MapReduce-based software frameworks, such as Hadoop (White, 2012) and Spark (Zaharia et al., 2012): after shuffling bundle data to the corresponding groups, each process gets its data in its bundle and then sends its data to its group aggregators, who write the bundle to PFS. Here, shuffling bundle data to the corresponding groups can be perceived as the first round of MapReduce, aggregation and write can be perceived as the second round of MapReduce. The difficulty of implementing BPAR using Hadoop/Spark on HPC systems lies in two aspects. First, most of existing HPC systems have their own resource managers, users have to go through special steps to enable Hadoop/Spark, such as Gordon at San Diego Supercomputer Center (SDSC) (SDSC Gordon User Guide, n.d.) and Wrangler at Texas Advanced Computing Center (TACC) (WRANGLER, n.d.). Second, many HPC programs are written by C and MPI, it is nontrivial to support these programs using traditional Hadoop (Java-based)/Spark (Scala-based), though there are already some ongoing work to provide these supports (Lu et al., 2014; Wang et al., 2014). Nonetheless, a general BPAR implementation using Hadoop/Spark still carries great potential if all the technical barriers are tackled.

The rest of this article presents three partitioning schemes under the BPAR framework. Unlike PnetCDF, all three schemes can maintain the existing file format of GEOS-5, while still delivering good performance. We experimentally compare their performance which will serve as guidelines for application practitioners to select the one that best fit their application's I/O workload and system configuration.

IV Representative partitioning strategies of BPAR

In this section, we present the three partitioning strategies and their individual advantages and disadvantages under the BPAR framework. Our study is based on GEOS-5. The layout of each 2-D plane is essentially a contiguous data extent within its bundle file and shared among all the processes. Such logical file layout among processes is most common in scientific workloads (Bent et al., 2009). The three partitioning strategies are developed from the perspectives of load balancing, data locality, and network congestion respectively.

A Balanced partitioning

The Balanced Partitioning Scheme (BPS) assigns the number of processes in each AG in proportion to the data size of each bundle. The rationale behind BPS is to balance the workload in each AG according to equation (2), thus minimizing the I/O time of the stragglers. For a data set with B bundles, let the data size of bundle i be Si, the total number of processes be n, the group size of AG_i be A_i, then A_i amounts to:

$$A_i = (n - B) \times \left(S_i / \sum_{i=1}^{B} S_i \right) + 1 \qquad (3)$$

Equation (3) first subtracts B from n to reserve 1 process for each bundle, and this 1 process is added to the end. The rest $n - B$ processes are assigned to each AG proportionally to the size of the corresponding bundle. When the number of processes is smaller than the number of bundles, a process can take charge of I/O for multiple bundles. Bundles are assigned to processes in a round-robin manner.

For each bundle, BPS evenly assigns the data planes to each process. The process that takes charge of a data plane is named the plane root. It gathers the entire plane data from the other processes. A master process is selected as the bundle root to gather the data from all of the plane roots and write the data to storage.

Figure 6 details such procedures. Suppose Bundle1 and Bundle2 are the two bundles involved in the I/O operation. Var1 and Var2 are a 2-D variable and a 3-D variable in Bundle1, respectively, and Var3 is a 2-D variable in Bundle2. Initially, each plane is shared by four processes. Following the BPS scheme, three processes are assigned to AG1, which take charge of Bundle1's I/O. One process is assigned to AG2, which

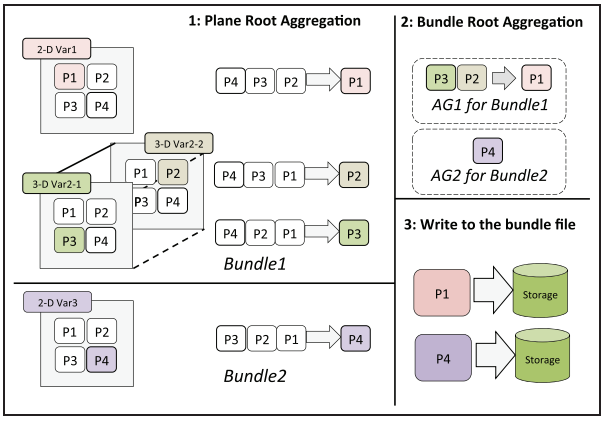

Figure 6. Write two bundle files using BPS. BPS: Balanced Partitioning Scheme.

takes charge of Bundle2's I/O. Inside Bundle1, three processes serve as the plane root for the three planes. Bundle2 has only one plane, and the plane root is the process P4. In the first stage, processes shuffle data so that the plane root acquires all the data of their planes. After this step, the entire bundle data falls in its own AG. Then the plane root in both AGs sends the data to the bundle root which in turn receives the plane data and writes it to the storage.

BPS performs well since it balances the workloads among processes. However, it may not work well in some cases. For example, the smallest unit assigned to each bundle is a process, so processes within the same node may scatter over multiple AGs, requiring more internode communication. Therefore, a locality-oriented partition scheme is considered.

B Locality-oriented partitioning

The Locality-Oriented Partitioning Scheme (LPS) assigns processes to AGs at the unit of physical node. Supposing each physical node hosts p processes, using the same notation as BPS, the group size of AG_i can be approximated as:

$$A_i = (n - B \times p) \times \left(S_i / \sum_{i=1}^{B} S_i \right) + p \qquad (4)$$

Similar to BPS, LPS reserves one physical node for each bundle, which hosts $B \times p$ processes. By subtracting $B \times p$ from n and adding p to the end, equation (4) makes sure that there is at least one physical node for each bundle. A_i is finally set to the closest number that is 1 or a multiple of p to make sure the smallest unit assigned to each bundle is a physical node. When the number of nodes is smaller than the number of bundles, a node can take charge of I/O for multiple bundles. Bundles are assigned to nodes in a round-robin manner. After deciding the AG for each bundle, LPS follows the same step as BPS.

Compared with BPS, the larger partitioning unit in LPS achieves augmented locality inside each AG and higher parallelism among different AGs. Nonetheless, the promoted locality and parallelism is attained at the expense of increased imbalance of workload distribution. For instance, for a workload that includes many small bundles that actually require fewer than p processes, the overprovisioned processes should be better assigned to those larger bundles who are potential stragglers. In addition, for scientific applications whose data are unevenly decomposed, there will be more space for locality-oriented optimization. For instance, we can assign a node to the bundle that contains most of this node's data to further reduce internode communication. How to assign nodes to bundles for these applications using LPS require specific information of how data are decomposed initially, which is outside our scope of discussion.

C Isolation-driven partitioning

In both BPS and LPS, the plane root is evenly selected among the AG members for each plane in the same bundle. Therefore, every process can be the plane root if the number of planes is larger than the number of processes. On the other hand, each plane is initially shared by all the processes, so every plane root needs to receive its data from all the other processes during data shuffling. Therefore, the shuffling operation is essentially an all-to-all communication that each process needs to send and receive data from all the other processes concurrently. Such communication pattern can lead to heavy network congestion among processes, which prolongs $T_{sf\,1}$ in equation (1).

Isolation-Driven Partitioning Scheme (IPS) is introduced to alleviate such congestion. IPS takes the same group partitioning scheme as BPS (see equation (3)) but differs by isolating the traffic to each process. The key idea of IPS is to have each process receive the data only from a small portion of corresponding processes rather than from all the processes. To achieve this purpose, each plane is no longer gathered entirely by the plane root during data shuffling. Instead, it is gathered by all the members in the same AG, and each member in this AG gather the plane data from its corresponding processes in other AGs. For instance, in Figure 7, Bundle1, Bundle2, and Bundle3 respectively, possess 5, 3, and 1 plane(s). These planes are shared among nine processes. Following IPS, the group sizes of AG1, AG2, and AG3 are proportionally set to 5, 3, and 1. Then, for each plane of Bundle1, processes P1-P4 receive the plane data respectively from P6-P9. P6-P9 are the remaining processes belonging to other groups. In this way, each process only receives data from one other process. Similar operations happen to Bundle2 and Bundle3.

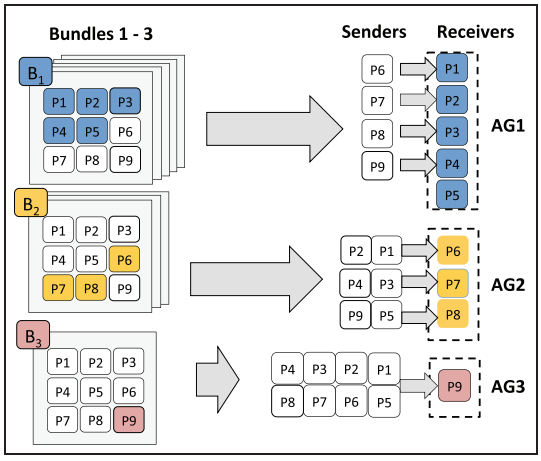

Figure 7. Partitioning scheme of IPS. IPS: Isolation-Driven Partitioning Scheme.

Instead, if we follow BPS or LPS, every process needs to receive the plane data from all other eight processes during data shuffling for its own plane.

Like BPS and LPS, inside each AG, a bundle root is elected to gather the plane data and write the data to storage. For brevity, such procedure is not shown in Figure 7. IPS is able to reduce T_{sfl} by alleviated congestion compared to the all-to-all data shuffling in BPS and LPS.

V Experimental evaluation

In this section, we systematically evaluate the effectiveness of BPAR. The results are gathered from the Discover supercomputer (Discover Supercomputer Statistics, n.d.) that is operated by the NASA Center for Climate Simulation (NASA Center for Climate Simulation, n.d.). The Discover supercomputer has an aggregate of 67 racks, containing 43,240 compute cores that yields 1.0018 Pflops/s of computational power. Each compute node consists of either two 2.6 GHZ, oct-core Intel Xeon Sandy Bridge processors with 32 GB of memory or two 2.8 GHZ, and hex-core Intel Xeon Westmere processors with 24 GB of memory. The underlying storage system of the Discover supercomputer uses the IBM GPFS and consists of 2.46 PB of total storage.

For our evaluation, we place eight processes on each physical node. We run GEOS-5 to simulate 24-h climate change with a bundle output frequency of 3 h. This results in a total of 56 files for the 8-time step run. After each simulation, the average I/O time of each time step is calculated. We run each simulation five times and get the median for the result.

A Overall performance of BPAR

To evaluate the performance of BPAR, we implement BPAR on top of GEOS-5 with its three partitioning schemes. Figure 8 shows its total I/O time as a result of the increasing number of processes.

The BPS, LPS, and IPS demonstrate compelling benefits over GEOS5-NetCDF. They reduce the I/O time delivered by GEOS5-NetCDF by 53%, 47%, and 51% on average, respectively. In addition, all three cases show better scalability. This is because unlike baseline NetCDF-based GEOS-5, which couples all the processes in writing each bundle, BPAR partitions the processes into different AGs, thereby enabling all of the AGs to work concurrently. Meanwhile, the smaller communication domain in each AG alleviates the contention, which is the main reason for the more durable scalability.

We have also observed distinct performance patterns between these three schemes. Although IPS initially performs the worst, it gradually catches up and ultimately delivers the optimal performance at 256 and 512 processes. On the contrary, BPS initially performs the best but lags behind when compared with IPS at 256 and 512 processes. Although LPS initially yields comparable performance to BPS and better performance than IPS, such trend stops at 256 processes, and it performs the worst from this point on.

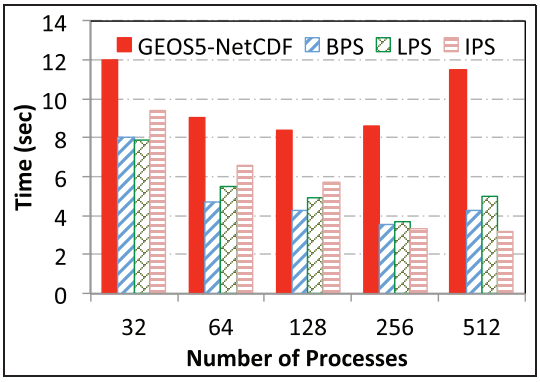

Figure 8. Performance of GEOS-5 with BPAR. GEOS-5: Goddard Earth Observing System, version 5; BPAR: Bundle-based PARallel Aggregation.

B An in-depth understanding of BPAR's performance

To better understand the distinct performance of BPAR's three partitioning schemes, we analyze the major overhead dominating the I/O time. As mentioned earlier, the total I/O time of BPAR is mainly composed of shuffle time, aggregation time, and write time. Since the aggregation and write operation are interleaved with each other, we name the entirety as collective write operation.

1 Analysis of shuffle operation. Figure 9 shows the shuffle time of BPS, LPS, and IPS. Among the three schemes, IPS constantly delivers the optimal performance, it also scales the best. This is because the all-to-all shuffling in BPS and LPS results in heavy network congestion. IPS alleviates the network congestion by restricting the incoming network traffic of each process from the other processes to only a portion of the corresponding processes. In contrast, LPS almost always consumes the most shuffle time. This is because LPS prioritizes locality over the balanced workload. By assigning the processes in the same physical node to the same AG, some of the smallest AGs acquire the overprovisioned number of processes, while other AGs that actually starve for more resources are left unnoticed, resulting in the prolonged overall shuffle time.

Figure 10 shows the data shuffling time spent on the largest bundle named "moist." As expected, LPS consumes the most shuffle time on this bundle. Table 1

Table 1. Group size of moist.

Number of processes	32	64	128	256	512
BPS	11	21	39	79	155
LPS	8	8	8	72	136
IPS	11	21	39	79	155

BPS: Balanced Partitioning Scheme; IPS: Isolation-Driven Partitioning Scheme; LPS: Locality-Oriented Partitioning Scheme.

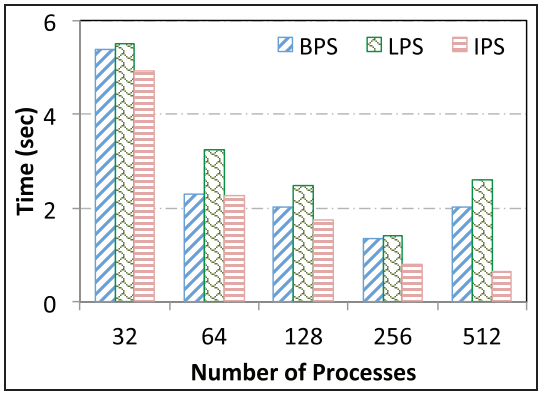

Figure 9. Total shuffle time of BPS, LPS, and IPS. BPS: Balanced Partitioning Scheme; IPS: Isolation-Driven Partitioning Scheme; LPS: Locality-Oriented Partitioning Scheme.

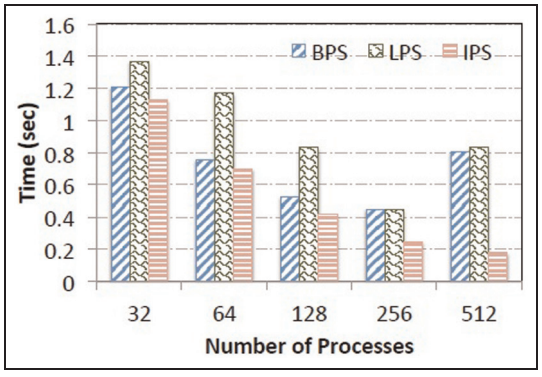

Figure 10. Comparison of the shuffle time of moist.

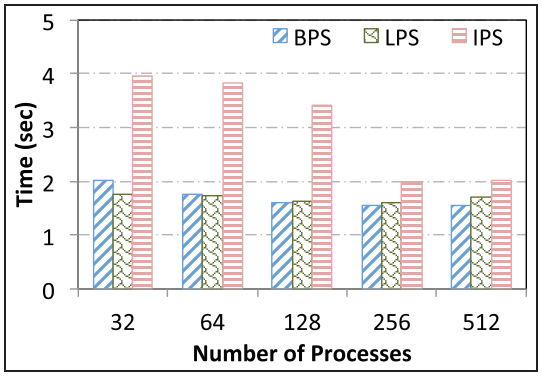

Figure 11. The performance of collective write operation.

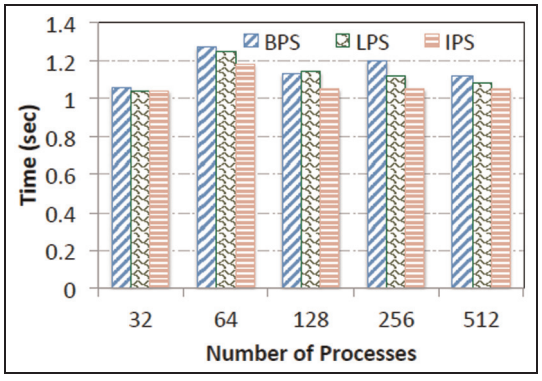

Figure 12. Comparison of the write time of moist.

further reveals the group size of LPS is constantly the smallest. This explains the trends in Figures 9 and 10.

2 Analysis of collective write operation. We have also measured the time spent on collective write operation as shown in Figure 11. Overall, collective write time of all three schemes initially decreases with the growing number of processes, then cease to decrease at a large number of processes. This is because each physical node on Discover is able to absorb the incoming stream from the processes on multiple nodes (each node launches eight processes); when it is saturated, the performance is slowed down by the heightened contention. In addition, we observe the collective write time of IPS is significantly beyond the other two. The collective write time of LPS and BPS are close to each other. To find the reason, we respectively calculate the aggregation time and the write time of the bundle roots.

Figure 12 reveals the write time of the three schemes for bundle moist. Overall, the write time of BPS, LPS, and IPS are close to each other. This is because the write operation is handled by the bundle root for all three schemes. Therefore, the different aggregation time is the major reason for the distinct collective write time. As we can see from Figure 13, the aggregation time of LPS is similar to BPS, despite the better locality of LPS. This is because the high-speed Infiniband deployed on Discover blurs the performance gap

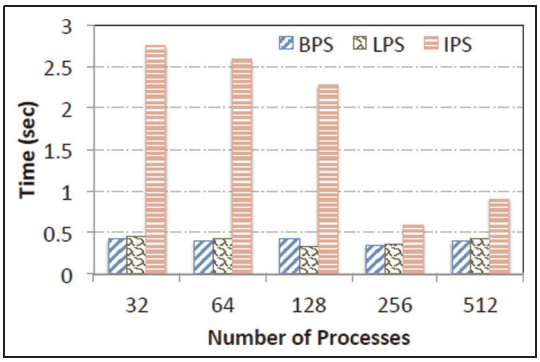

Figure 13. Comparison of the aggregation time of moist.

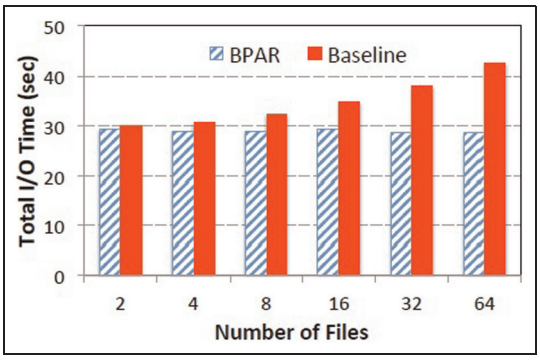

Figure 14. Comparison of the total I/O time on Discover.

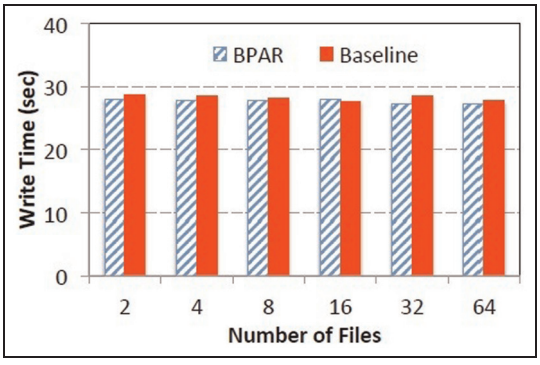

Figure 15. Comparison of the write time on Discover.

between intranode communication and internode communication. In particular, intranode communication in LPS is achieved by memory copy among different processes, the QDR Infiniband on Discover delivers comparable network bandwidth to memory bandwidth. So the more intranode communication in LPS shows no more advantage over BPS. We have also observed the aggregation time of IPS is much longer than BPS and LPS. Such large aggregation time results from the transfer size of IPS. IPS evenly distributes the plane data to group members, so the transfer size is a fraction of the plane size, while the transfer size of BPS and LPS amount to the plane size.

C Analysis of BPAR's support for parallel I/O outside GEOS-5

We have also conducted experiments to investigate the benefit that BPAR brings to collective I/O for the access of multiple files. In the baseline case, all the processes write to multiple files following ROMIO, a collective MPI-IO implementation that have all the processes collaboratively shuffle and write the data belonging to one file each time. In the BPAR case, all the processes write multiple files following BPAR. Since the two major operations involved in ROMIO are shuffle and I/O, for fair comparison, we include the same two operations in BPAR without the aggregation phase. Namely, after each AG acquires its file data, it directly writes the file to the file system without aggregation. We use BPS to assign the processes into each AG. To reflect the effect of BPAR on different PFSs, we characterize its performance on both Discover (Discover Supercomputer Statistics, n.d.) and Titan (Introducing Titan, n.d.) supercomputer. In all the rest of the experiments, we place one process on each physical node.

1 Experimental analysis on discover. We first evaluate BPAR support on Discover supercomputer, we have 64 processes (64 is the maximum allowable number of physical nodes allocated to an application on Discover) write a varying number of shared files to the PFS. Each file is chopped into 16 KB segments, and these segments are assigned to all the processes in a round-robin manner. The total data size written by each process is 1 GB, and all the shared files are of the same size. Figure 14 compares the total I/O time of the baseline and BPAR. BPAR initially demonstrates only slight advantage over the baseline. However, its benefit becomes more and more obvious with an increasing number of shared files involved. On average, BPAR accounts for 85% of the baseline's I/O time.

To further investigate the main factors that contribute to BPAR's performance, we have dissected the write time and shuffle time of the baseline and BPAR, shown in Figures 15 and 16 respectively. The write time of the BPAR is comparable to the baseline. This is because files in GPFS are striped across all disks in the file system (Schmuck and Haskin, 2002). Both technique utilizes all the disks in writing each file. On the other hand, the shuffle time of BPAR stays constantly below the baseline, the gap is prolonged with the more and more files involved. This is because ROMIO couples all the processes together in writing each file, so writing multiple files involves multiple rounds of shuffle and write operations. Since the shuffle time of each round is bounded by the last process that receives its entire data set, the accumulated shuffle time over multiple-round operations are significantly prolonged by the slowest

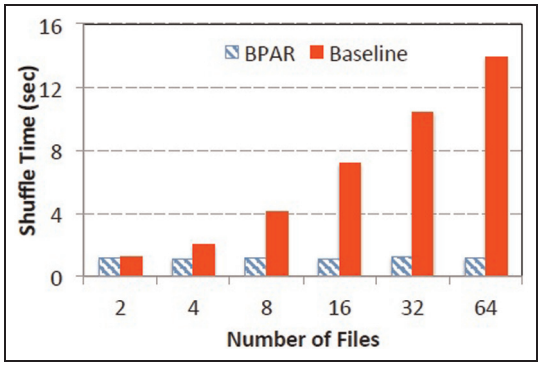

Figure 16. Comparison of the shuffle time on Discover.

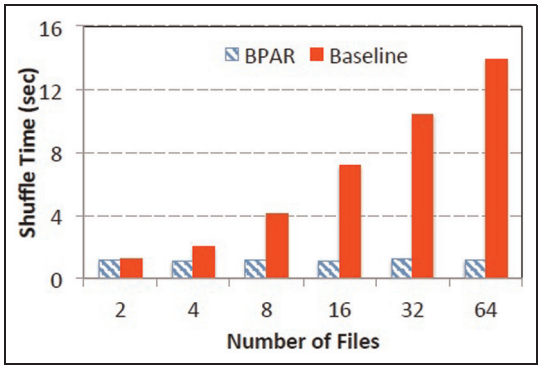

Figure 17. Comparison of the total I/O time on Titan.

processes in each round. In contrast, BPAR involves only one round of data shuffle to send each file to its corresponding AG. Although its shuffle time is also bounded by the slowest process, such impact is alleviated with its one-round operation.

2 Experimental analysis on Titan. To investigate BPAR's support on other modern PFSs, we have also compared the performance of the BPAR and the baseline on Titan following the same experiment set-up as the Discover.

Titan is the world's second largest supercomputer hosted in Oak Ridge National Laboratory (ORNL, Oak Ridge, Tennessee, USA). Each node is equipped with a 16-core 2.2 GHz AMD Opteron 6274 (Interlagos) processor, 32 GB of RAM, and a connection to the Cray custom high-speed interconnect. Titan is connected to Spider II, a center-wide Lustre-based file system. It features 30 PB of disk space, offering 1 TB/s aggregated bandwidth organized in two non-overlapping identical file systems, each providing 500 GB/s I/O performance. The default stripe size of each created file is 1 MB. The default stripe count is 4.

Figure 17 compares the total I/O time of BPAR and the baseline. On average, the I/O time of BPAR accounts for only 20% of the baseline, and the gap between the two is aggrandized with more and more files involved. Figures 18 and 19 respectively reveal the write time and shuffle time of the two. It can be perceived that the performance gap mainly comes from the write operation. This is because unlike the GPFS that stripes each file across all the disks, Lustre file system strives to stripe each file across a set of distinct disks. By shuffling each file to its corresponding AG, all AGs are able to concurrently work on their own files, thereby utilizing the supplied bandwidth of all the assigned disks. In contrast, ROMIO couples all the processes in writing each file, so in each round only the disks belonging to one file is utilized. The more files involved, the more percent of supplied bandwidth is wasted. We have also observed that the data shuffle

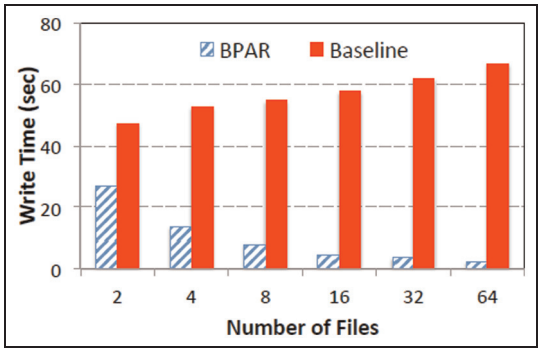

Figure 18. Comparison of the write time on Titan.

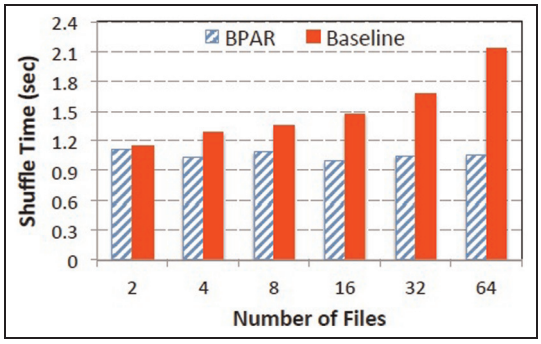

Figure 19. Comparison of the shuffle time on Titan.

benefit of BPAR on Titan is less significant than Discover. This is because each process on Titan supercomputer is able to absorb more traffic than the Discover supercomputer. Therefore, the bandwidth supplied to each shuffler is less likely to be impacted by other competing shufflers. To demonstrate such effect, we have from 1 to 32 processes located on different physical nodes concurrently send 32 GB network requests to 1 receiving process located on another nodes and compare the ingress bandwidth on Discover and Titan, the result of which is shown in Figure 20. Both bandwidths are saturated with two processes. The

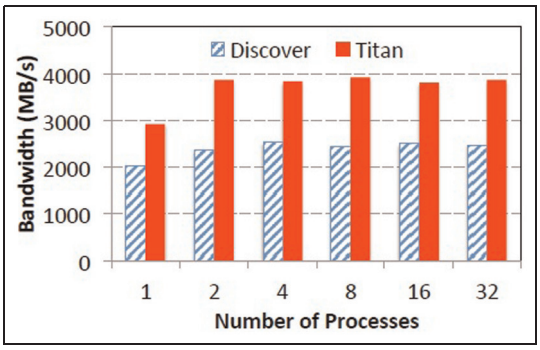

Figure 20. Comparison of the ingress bandwidth per process.

ingress bandwidth of Titan is 54% more than Discover on average.

D Discussion

From the experimental analysis, we find GEOS-5 with BPAR delivers significantly higher throughput than baseline GEOS-5, because BPAR's partitioned and decoupled I/O on each bundle file results in better parallelism and scalability. Meanwhile, we perceive different performance between BPS, LPS, and IPS. While IPS reduces the shuffle time with mitigated network congestion, it is likely to deliver suboptimal performance in aggregation operations because the small transfer size is not favored by the network. This indicates that IPS is better used for workloads with large plane sizes. More generally, it fits the workload in which the shared file is composed of several large, contiguous extents striped among each process. BPS balances the workload of each AG well, so it delivers the most constant performance compared with the other two. BPS can serve as the default scheme for BPAR with a reasonable number of processes, but when the number of processes is large, BPS is better replaced by IPS due to the heavy congestion incurred by data shuffling. LPS achieves good locality, but it may result in an imbalanced workload. This is especially true for workloads that involve a lot of small bundles. Meanwhile, the locality is not necessary for the system deployed with high-speed network, such as Infiniband. Ideally LPS should be used on the system whose network bandwidth is restricted. The application developer should be aware of the workload distribution under LPS.

Our characterization of BPAR on Discover and Titan supercomputing reveals two of its benefits: First, BPAR makes better utilization of disks by having multiple groups concurrently work on their own files. Second, by decoupling the collective write on each file, BPAR can parallelize the operation on multiple files, reducing the synchronization overhead. Our characterization does not include the operation of aggregation. We believe BPAR can achieve even more compelling performance improvement via wise aggregation. Since the number of aggregators elected can pose significant impact to the I/O bandwidth (Yu et al., 2008). When the aggregation is included, we believe the BPS, LPS, and IPS are applicable according to their respective advantages.

VI Related work

Improving the I/O performance on large-scale HPC systems has been a highly active research topic and has gained broad attention over the past few decades. In general, work surrounding such topic can be categorized into three levels: file sytem-level optimizations, middleware-level optimizations, and application-level optimizations.

Early work on file systems includes the large-scale development and adoption of PFSs, such as Lustre (Liu et al., 2013c), GPFS (Schmuck and Haskin, 2002), Parallel Virtual File System (Ross et al., 2000) and the effort to optimize their internal implementations by augmenting network transfers (Carns et al., 2009), caching strategies (Byna et al., 2008), I/O scheduler (Qian et al., 2009), or through more hardware-based integrations and upgrades (Oral et al., 2013, 2014; Shipman et al., 2012). These works focus on the fundamental software layers that directly interact with the storage, which is orthogonal to our work. Recently, HPC systems are experiencing a paradigm shift from compute-centric paradigm to data-centric paradigm (Wang et al., 2014a). Many HPC systems have provided support for MapReduce-based data analytics, such as Gordon on SDSC and Wrangler at TACC. The aggregated local disk I/O with HDFS can deliver scientific applications with scalable I/O throughput. However, unlike conventional PFSs in HPC, Hadoop Distributed File System (HDFS) is an append-only file system designed for batch processing, it doesn't support some of the typical scientific workloads, such as N-1 I/O pattern (multiple processes access a shared file).

The middleware-level optimizations largely center on parallel I/O techniques and I/O off-loading. MPI-IO (Thakur et al., 1999) is a parallel I/O middleware widely applied to scientific applications. It provides applications with parallel access to shared data sets, in addition to superior aggregation strategies to coalesce small data into larger ones. Advanced parallel I/O middleware libraries such as PnetCDF (Li et al., 2003), HDF5 (HDF5 Home Page, n.d.) are built on top of MPI-IO, while inheriting most of its features, they allow applications to access the shared data set at the granularity of variables, which are more user-friendly. The popularity of MPI-IO and its ramifications have drawn plenty of effort for their optimizations (Gao et al., 2009; Howison, 2012; Liu, et al., 2013a; Xu et al.,

2013). However, these techniques are not feasible to frequent and small collective I/O workload since the synchronization overhead and I/O contention caused by the small I/O requests will soon submerge the benefit from enhanced parallelism (Yu and Vetter, 2008).

Meanwhile, a plethora of other powerful parallel techniques are designed to more or less compensate for the deficiency of MPI-IO and its derivatives. ADIOS (Lofstead et al., 2009) uses chunking to improve the data locality and the request size. Meanwhile, it alleviates the synchronization overhead by allowing users to predefine the data format in XML file. However, this introduces extra overhead for the users, and the output in Binary Packed format is only accessible using the ADIOS interface. PLFS (Bent et al., 2009) improves the performance by transforming the one-file-multiple-processes (N-1) pattern to one-file-per-process (N-N) pattern. In doing so, it maximizes the I/O sequentiality and reduces the synchronization overhead while still retaining high concurrency. Like ADIOS, the output format is only recognizable by PLFS interface and in some PFSs, the N-N pattern can introduce nonnegligible metadata overhead (Yu et al., 2008).

Aside from parallel I/O techniques, I/O off-loading is another extensively used middleware-level approach that aims to reduce both I/O workload and I/O time on the compute node. It achieves its purpose using dedicated I/O nodes. In general, it falls into three categories: I/O forwarding (Ali et al., 2009; Vishwanath et al., 2010), I/O staging (Abbasi et al., 2010; Nisar et al., 2008) and burst buffer (Liu et al., 2012; Wang et al., 2014c, 2015). I/O forwarding focuses on eliminating system noise from I/O operations by off-loading I/O to dedicated I/O nodes. Most notably, it has been applied on Blue Gene/P systems. I/O staging stages the data set to a set of dedicated I/O nodes for online data sharing and analysis. Burst buffer is a recent technique that captures the bursty behavior of scientific applications. Burst buffer system can temporarily buffer the burst of scientific dataset in the high-performance storage such as dynamic RAM and solid-state disk and allows the actual data flushing to the file system to be conducted simultaneously with application's ensuing computation, thereby largely reduce the time spent on scientific applications' I/O phase.

The application-level optimizations generally focus on how to port the parallel techniques to the applications for enhanced parallelism. Tian et al. (2013) add ADIOS support to GEOS-5 and obtain significant I/O performance improvement. Li et al. (2002) replace the sequential I/O operation in adaptive mesh refinement cosmology application with MPI-IO and HDF5 and point out the advantages and disadvantages of these techniques. Similarly, Johnson and Bethune, 2012 optimize PARA-BMU, the solver part of a voxel-based bone modeling suite using NetCDF and HDF5 libraries.

Our work stays between the middleware-level and application-level optimization. Different from the aforementioned work, our work captures the scientific applications' behavior that in each I/O phase there will be multiple bundles files generated and stored on the PFS (Wang et al., 2014d). We focus on researching the potential benefit that can be attained from cross-bundle optimizations and build the related framework to accelerate the representative application GEOS-5, together with a broad range of other scientific applications on various computing platforms. BPAR mitigates several aforementioned deficiencies of existing parallel I/O techniques, such as low storage utilization, synchronization overhead, I/O contention, and so on.

VII Conclusions

In this work, we have explored the major factors that restrict the I/O performance of baseline NetCDF-based GEOS-5 I/O and GEOS-5 with PnetCDF. Based on our analysis, we research a BPAR framework, together with three of its partitioning schemes to enhance the I/O performance of GEOS-5 and a broad range of other scientific applications. Our experiment result reveals BPAR can outperform baseline GEOS-5 by up to 2.1×, and it is promising for I/O acceleration on the different computing platforms.

As a future work, we plan to research a smart aggregation scheme that is able to adaptively select the best number of aggregators for each AG based on the workload and plug this scheme into BPAR framework. We will also implement such framework inside the MPI-IO library to enhance its support for multiple-file operations.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work is funded in part by a NASA grant NNX11AR20G and enabled by the U.S. National Science Foundation award CNS-1059376.

References

Abbasi H, Wolf M, Eisenhauer G, et al. (2010) DataStager: scalable data staging services for petascale applications. *Cluster Computing* 13(3): 277–290.

Ali N, Carns P, Iskra K, et al. (2009) Scalable I/O forwarding framework for high-performance computing systems. In: *IEEE International Conference on Cluster Computing and Workshops, 2009 (CLUSTER'09)*, New Orleans, LA, 31 August–4 September 2009, pp. 1–10.

Bent J, Gibson G, Grider G, et al. (2009) PLFS: a checkpoint filesystem for parallel applications. In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Portland, OR, 14–20 November 2009. Portland: ACM, pp. 1–12.

Byna S, Chen Y, Sun X-H, et al. (2008) Parallel I/O prefetching using MPI file caching and I/O signatures. In: *International Conference for High Performance Computing, Networking, Storage and Analysis, 2008 (SC 2008)*, Austin, TX, 15–21 November 2008. Piscataway, NJ: IEEE, pp. 1–12.

Carns P, Lang S, Ross R, et al. (2009) Small-file access in parallel file systems. In: *IEEE International Symposium on Parallel & Distributed Processing, 2009 (IPDPS 2009)*, Rome, 23–29 May 2009, pp. 1–11.

Corbett P, Feitelson D, Fineberg S, et al. (1996) Overview of the MPI-IO parallel I/O interface. In: Jain R, Werth J and Browne JC (eds.) *Input/Output in Parallel and Distributed Computer Systems*. Springer, pp. 127–146.

Dean J and Ghemawat S (2008) "MapReduce: simplified data processing on large clusters". In *Communications of the ACM*. Vol. 51, January 2008. New York: ACM, pp. 107–113.

Discover Supercomputer Statistics (n.d.) http://www.nccs.nasa.gov/discoverfront.html (accessed 26 February 2014).

Gao K, Liao W.-K, Choudhary A, et al. (2009) Combining I/O operations for multiple array variables in parallel netCDF. In: *IEEE International Conference on Cluster Computing and Workshops, 2009 (CLUSTER'09)*, New Orleans, LA, 31 August–4 September 2009. New Orleans, LA: IEEE, pp. 1–10.

HDF5 Home Page (n.d.) http://www.hdfgroup.org/HDF5/ (accessed 27 January 2014).

Howison M (2012) Tuning HDF5 for Lustre file systems. In: *Proceedings of the Workshop on Interface and Abstractions for Scientific Data Storage (IASDS10)*, Heraklion, Crete, Greece, 24 September 2010.

Introducing Titan (n.d.) http://www.olcf.ornl.gov/titan/ (accessed 9 January 2014).

Johnson N and Bethune I (2012) Adding parallel I/O to PARA-BMU. Academic Research Training and Support.

Li J, Liao W.-K, Choudhary A, et al. (2002) I/O analysis and optimization for an AMR cosmology application. In: *IEEE International Conference on Cluster Computing, 2002*, Chicago, September 2002, IEEE, pp. 119–126.

Li J, Liao W.-K, Choudhary A, et al. (2003) Parallel netCDF: A high-performance scientific I/O interface. In: *ACM/IEEE Conference on Supercomputing, 2003*, Phoenix, AZ, 15–21 November 2003. Phoenix, AZ: IEEE, p. 39.

Liu N, Cope J, Carns P, et al. (2012) On the role of burst buffers in leadership-class storage systems. In: *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, Pacific Grove, CA, 16–20 April 2012. Pacific Grove, CA: IEEE, pp. 1–11.

Liu J, Crysler B, Lu Y, et al. (2013a) Locality-driven high-level I/O aggregation for processing scientific datasets. In: *2013 IEEE International Conference on Big Data*, Silicon Valley, CA, 6–9 October 2013. Santa Clara: IEEE, pp. 103–111.

Liu Z, Lofstead J, Wang T, et al. (2013b) A case of system-wide power management for scientific applications. In: *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, Indianapolis, IN, 23–27 September 2013, pp. 1–8.

Liu Z, Wang B, Wang T, et al. (2013c) Profiling and improving I/O performance of a large-scale climate scientific application. In: *International Conference on Computer Communications and Networks (ICCCN)*, Nassau, Bahamas, July 2013. IEEE, pp. 1–7.

Lofstead J, Zheng F, Klasky S, et al. (2009) Adaptable, metadata rich IO methods for portable high performance IO. In: *IEEE International Symposium on Parallel & Distributed Processing, 2009 (IPDPS)*, Rome, 23–29 May 2009. Rome: IEEE, pp. 1–10.

Lu X, Liang F, Wang B, et al. (2014) DataMPI: extending MPI to hadoop-like big data computing. In: *IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS'14)*, Phoenix, AZ, 19–23 May 2014. Phoenix, AZ: IEEE, pp. 829–838.

NASA Center for Climate Simulation (n.d.) http://www.nccs.nasa.gov/index.html (accessed 26 February 2014).

Nisar A, Liao W.-k and Choudhary A (2008) Scaling parallel I/O performance through I/O delegate and caching system. In: *International Conference for High Performance Computing, Networking, Storage and Analysis, 2008 (SC 2008)*, Austin, TX, 15–21 November 2008. Austin, TX: IEEE, pp. 1–12.

Oral S, Dillow DA, Fuller D, et al. (2013) OLCF's 1 TB/s, next-generation Lustre file system. In: *Proceedings of the Cray User Group Conference, 2013 (CUG)*, Napa Valley, CA, May 2013.

Oral S, Simmons J, Hill J, et al. (2014) Best practices and lessons learned from deploying and operating large-scale data-centric parallel file systems. In: *International Conference for High Performance Computing, Networking, Storage and Analysis, (SC14)*, New Orleans, LA, 16–21 November 2014. New Orleans, LA: ACM, pp. 217–228.

Qian Y, Barton E, Wang T, et al. (2009) A novel network request scheduler for a large scale storage system, *Computer Science-Research and Development* 23: 143–148.

Ross RB, Thakur R, Carns PH, et al. (2000) PVFS: A parallel file system for Linux clusters. In: *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta, GA, October 2000. Atlanta, GA: USENIX, pp. 317–327.

Sankaran R, Mellor-Crummy J, DeVries M, et al. (2008). *Terascale Direct Numerical Simulations of Turbulent Combustion Using S3D: Technical Report*. Livermore, California, USA: Sandia National Laboratories.

Sarawagi S and Stonebraker M (1994) Efficient organization of large multidimensional arrays. In: *Proceedings 10th International Conference Data Engineering, 1994 (ICDE)*, Houston, TX, 14–18 February 1994. Houston: IEEE, pp. 328–336.

Schmuck FB and Haskin RL (2002) GPFS: A shared-disk file system for large computing clusters. In: *Proceedings of the FAST'02 Conference on File and Storage Technologies*, Monterey, CA, January 2002. Berkeley, CA: USENIX, pp. 231–244.

SDSC Gordon User Guide (n.d.). Available at: https://portal.xsede.org/sdsc-gordon/ (accessed 26 February 2014).

Shipman GM, Dillow DA, Fuller D, et al. (2012) A next-generation parallel file system environment for the OLCF. In: *Cray User Group Conference (CUG)*, Stuttgart, Germany, 29 April–3 May 2012.

Thakur R, Gropp W and Lusk E (1999) Data sieving and collective I/O in ROMIO. In: *The Seventh Symposium on the Frontiers of Massively Parallel Computation*, Annapolis, MD, 21–25 February 1999. McLean, VA: IEEE, pp. 182–189.

Tian Y, Liu Z, Klasky S, et al. (2013) A lightweight I/O scheme to facilitate spatial and temporal queries of scientific data analytics. In: *IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, Long Beach, CA, 6–10 May 2013. Lake Arrowhead, CA: IEEE, pp. 1–10.

Vishwanath V, Hereld M, Iskra K, et al. (2010) Accelerating I/O forwarding in IBM Blue Gene/P systems. In: *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, New Orleans, LA, 13–19 November 2010, pp. 1–10.

Wang Y, Goldstone R, Yu W, et al. (2014a) Characterization and optimization of memory-resident mapreduce on HPC systems. In: *IEEE 28th International Parallel and Distributed Processing Symposium*, Phoenix, AZ, 19–23 May 2014, pp. 799–808.

Wang Y, Jiao Y, Xu C, et al. (2014b), Assessing the performance impact of high-speed interconnects on mapreduce. In: *Specifying Big Data Benchmarks Workshop*, Springer, pp. 148–163.

Wang T, Oral HS, Wang Y, et al. (2014c) Burstmem: a high-performance burst buffer system for scientific applications. In: *IEEE International Conference on Big Data (Big Data)*, Washington DC, 27–30 October 2014, pp. 71–79.

Wang T, Vasko K, Liu Z, et al. (2014d) BPAR: a bundle-based parallel aggregation framework for decoupled I/O execution. In: *2014 International Workshop on Data Intensive Scalable Computing Systems (DISCS)*, New Orleans, LA, 16 November 2014, pp. 25–32.

Wang T, Oral S, Pritchard M, et al. (2015) TRIO: burst buffer based I/O orchestration. In: *2015 IEEE International Conference on Cluster Computing (CLUSTER)*, Chicago, IL, 8–11 September 2015, pp. 1–10.

White T (2012) *Hadoop: the definitive guide*. Sebastopol, California: O'Reilly Media, Inc.

Wong P and der Wijngaart R (2003), NAS parallel benchmarks I/O version 2.4. Technical Report NAS-03-002NASA, Ames Research Center, Moffet Field, CA, USA.

WRANGLER (n.d.) https://www.tacc.utexas.edu/systems/wrangler/ (accessed 26 February 2014).

Xu C, Venkata MG, Graham RL, et al. (2013) SLOAVx: scalable LOgarithmic AlltoallV algorithm for hierarchical multicore systems. In: *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Delft, 13–16 May 2013, pp. 369–376.

Yu W and Vetter J (2008) Parcoll: partitioned collective I/O on the Cray XT. In: *37th International Conference on Parallel Processing, 2008 (ICPP'08)*, Portland, OR, 9–12 September 2008, pp. 562–569.

Yu W, Vetter JS and Oral HS (2008) Performance characterization and optimization of parallel I/O on the Cray XT. In: *IEEE International Symposium on Parallel and Distributed Processing, 2008 (IPDPS 2008)*, Miami, FL, 14–18 April 2008, pp. 1–11.

Zaharia M, Chowdhury M, Das T, et al. (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, 25–27 April 2012.

Author biographies

Teng Wang is currently a fourth-year PhD student at the Department of Computer Science in Florida State University. He received his master's degree in Software Engineering from Huazhong University of Science and Technology in 2012. He obtained his bachelor's degree in Computer Science from Zhengzhou University in 2009. His research interests include High Performance Computing, Parallel I/O, Storage Systems, and Cloud Computing.

Kevin Vasko is originally from Elberta, Alabama, and currently resides in Huntsville, Albama. In 2008, he obtained his Bachelor of Science in Computer Engineering from Auburn University. After graduation he worked as a software engineer at Neptune Technology Group until 2014. In 2015, he obtained his master of science in Software Engineering from Auburn University under the direction of Dr Weikuan Yu. He is currently working for Boeing Research & Technology in the Center for Applied Simulation and Analytics (CASA) as a software engineer.

Zhuo Liu received his PhD from the Department of Computer Science and Software Engineering at Auburn University in 2015 and received his bachelor's degree from Huazhong University of Science and Technology in 2007. His research interests include Cloud Computing, Big Data Analytics, Parallel I/O, and Storage Systems. He has joined Yahoo! as a software engineer after graduation.

Hui Chen worked on this project as a postdoctoral researcher at Auburn University and then joined Louisiana State University as a research assistant. He received his PhD and bachelor's degree from Beijing University of Posts and Telecommunications in 2012 and 2006, respectively. Before coming to United States, he has worked in Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences for 2 years. His research interests include Cloud Computing,

Energy Efficiency Management of Data Center, and Big Data Processing.

Weikuan Yu is an associate professor in the Department of Computer Science at Florida State University (FSU). Before joining FSU, he served as an assistant and associate professor at Auburn University until 2015. Prior to that, he was a research scientist at Oak Ridge National Laboratory (ORNL) until 2009. He received his PhD in Computer Science from the Ohio State University in 2006. He also holds a master's degree in Developmental Biology from the Ohio State University and a bachelor's degree in Genetics from Wuhan University, China. At FSU, he leads the Parallel Architecture and System Laboratory (PASL) for research and development on high-end computing, parallel and distributing networking, storage and file systems as well as interdisciplinary topics on computational biology. He is a senior member of The Institute of Electrical and Electronics Engineers.