2014 International Workshop on Data Intensive Scalable Computing Systems

BPAR: A Bundle-Based Parallel Aggregation Framework for Decoupled I/O Execution

Teng Wang Kevin Vasko Zhuo Liu Hui Chen Weikuan Yu
Auburn University, AL 36849
{tzw0019,vaskokj,zhuoliu,hchen,wkyu}@auburn.edu

***Abstract*—In today's "Big Data" era, developers have adopted I/O techniques such as MPI-IO, Parallel NetCDF and HDF5 to garner enough performance to manage the vast amount of data that scientific applications require. These I/O techniques offer parallel access to shared datasets and together with a set of optimizations such as data sieving and two-phase I/O to boost I/O throughput. While most of these techniques focus on optimizing the access pattern on a single file or file extent, few of these techniques consider cross-file I/O optimizations. This paper aims to explore the potential benefit from cross-file I/O aggregation. We propose a Bundle-based PARallel Aggregation framework (BPAR) and design three partitioning schemes under such framework that targets at improving the I/O performance of a mission-critical application GEOS-5, as well as a broad range of other scientific applications. The results of our experiments reveal that BPAR can achieve on average 2.1× performance improvement over the baseline GEOS-5.**

I. Introduction

Large-scale scientific applications can generate colossal multidimensional datasets during execution. This data is typically in the form of checkpoint-restart data, data analysis and visualization output. Such massive-scale datasets require commensurate I/O throughput to enable timely data analysis and spare memory space for the ensuing rounds of computation.

To shoulder the exploding data pressure, I/O techniques such as MPI-IO [1], PnetCDF [2] and HDF5 [3] have been designed to provide aid within this domain. These I/O techniques allow processes to access the shared dataset in parallel and offer a set of I/O optimizations such as data sieving [1], two-phase I/O [1] and chunking [4] to enhance the I/O performance.

Most of these techniques aim to optimize the access pattern on a single shared file or file extent. This is achieved by associating the interface with the file descriptor. 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 file to large, contiguous I/O requests. However, few of these I/O techniques consider cross-file optimizations which can lead to performance improvements for scientific applications as they generally access more than one file during their life cycle. For instance, the life cycle of the GEOS-5 application [5] is composed of several timesteps. In each timestep, it will generate multiple bundle files. Cross-file optimization allows processes to collaboratively work on all files concurrently, which extends the optimization scope and allows for higher potential performance improvement.

This paper aims to explore the potential benefit from cross-file 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 the writes on each bundle file to be conducted concurrently. However, the serial NetCDF employed by existing 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 writes 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 Parallel NetCDF (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 [2]. In addition, the original data format of GEOS-5 is not maintained by PnetCDF.

Therefore, a new solution that preserves the original GEOS-5's data format is needed with enhanced parallelism and reduced contention at scale. In this paper, we propose a Bundle-based PARallel Aggregation framework (BPAR) 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 experiments reveal that BPAR can achieve on average 2.1× performance improvement over the baseline GEOS-5.

The rest of this paper is organized as follows. Section II analyzes the I/O performance of the 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 and analyzes the effectiveness of the three partitioning schemes. Section VI reviews the related

978-1-4799-7038-4/14 $31.00 © 2014 IEEE
DOI 10.1109/DISCS.2014.6

work. Finally, we conclude the paper 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 performance of the baseline GEOS-5 and GEOS-5 with PnetCDF.

A. Overview of GEOS-5

GEOS-5 (Goddard Earth Observing System Model) is being developed by NASA to support the earth science research. It simulates climate changes that span diverse temporal granularities.

The simulation dataset is organized with the 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 datasets, 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 two-dimensional 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 Fig. 1. Bundle1 and Bundle2 are the two bundles written in the I/O phase, Bundle1 contains 2 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).

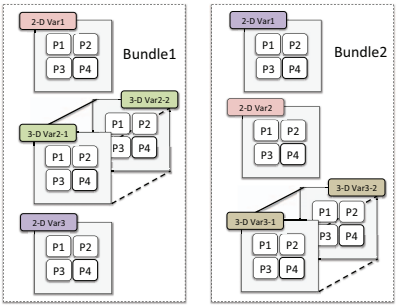

Fig. 1: The Data Organization of GEOS-5.

GEOS-5's life cycle alternates between computation phase and I/O phase. During each I/O phase the state variables are materialized on the underlying parallel filesystem (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. Original 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 materializes the plane on the filesystem, and proceeds to handle the next plane. So the I/O time is dominated by the communication 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 can not 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 dataset to the filesystem. 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 original NetCDF-based GEOS-5 and GEOS-5 with PnetCDF

To analyze the I/O performance of original NetCDF-based GEOS-5 and GEOS-5 with PnetCDF, we have systematically benchmarked their performance.

All the experiments are conducted on the Discover Supercomputer [6] that is operated by the NASA Center for Climate Simulation (NCCS) [7]. The Discover supercomputer has an aggregate of sixty-seven racks, containing 43,240 compute cores that yields 1.0018 Pflops/s of computational power. Each compute node consists of either a 2.6GHZ Intel Xeon Sandy Bridge processor with 32GB of memory or 2.8GHZ Intel Xeon Westmere processor with 24GB of memory. The underlying storage system of the Discover supercomputer uses the IBM GPFS and consists of 2.46PB of total storage.

For our analysis we place 8 processes on each physical node. We configure GEOS-5 to use 7 bundles of modeling data and simulate 24 hours with half-degree simulation, and a bundle output frequency for all bundles of 3 hours. This results in a total of 56 files for the 8-timesteps run. We run each experiment 5 times and get the median for the result.

Fig. 2 compares the two implementations in terms of the average I/O time on each timestep, these two implementations are denoted respectively by GEOS5-NetCDF and GEOS5-PnetCDF.

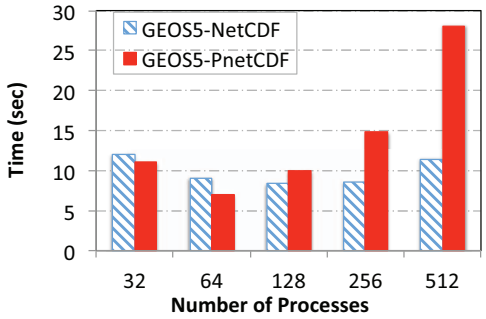

Fig. 2: The I/O Performance of original NetCDF-based GEOS-5 and GEOS-5 with PnetCDF.

As we can see in Fig. 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 original NetCDF-based GEOS-5 I/O firstly decreases from 32 to 128 processes then keeps on increasing for executions with processes more than 128. While the improved performance from 32 to 128 processes is attributed to the unsaturated ingress bandwidth of the master processes, this decline in performance with 128, 256, 512 processes results from the aggravated contention on the masters' limited resources.

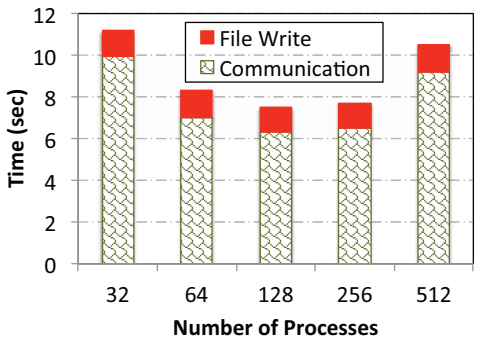

Fig. 3: Dissection of original NetCDF-based GEOS-5 I/O.

We further dissect the I/O time for a more elaborate analysis. Fig. 3 reveals the detailed time dissection of original NetCDF-based GEOS-5 I/O. The I/O time is dominated by the communication time. It first decreases and then is prolonged as the number of processes increases. This trend is consistent with Fig 2; In contrast, the write time is negligible due to the high aggregated I/O throughput rendered by GPFS on Discover.

The dissection of GEOS-5 with PnetCDF is shown in Fig. 4. It can be observed the two primary I/O operations (file write and file create) both impose nonnegligible overhead. When the number of processes increases, the file write time

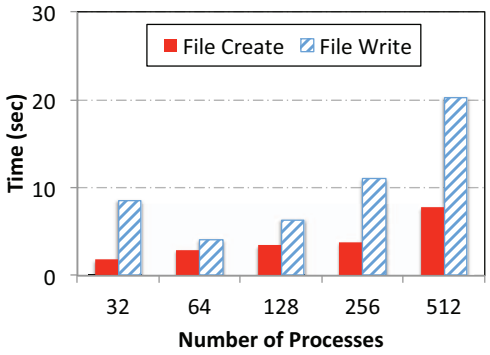

Fig. 4: Dissection of GEOS-5 with PnetCDF.

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 timestep. 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 original NetCDF-based GEOS-5 and GEOS-5 with PnetCDF have multiple drawbacks. original 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 and other advanced parallel I/O techniques.

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 original 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.

BPAR can be applied to a wide range of scientific applications that output several bundle files across I/O phases or inside each phase. 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 original NetCDF-based GEOS-5 I/O and PnetCDF.

Fig. 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 is initially distributed among all the four processes. According to a certain partitioning technique, the four processes are divided into two Aggregation Groups (AG), 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, e.g., 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 this figure), 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.

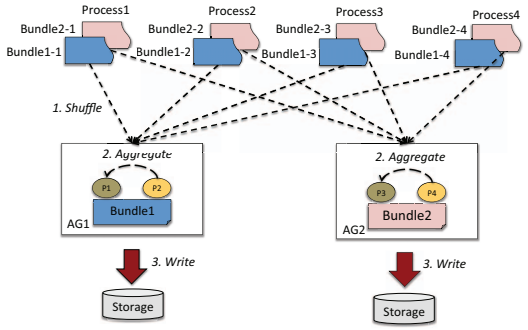

Fig. 5: A High-Level Overview of BPAR.

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 as:

$$T_i = T_{sfl} + T_{agg} + T_w \tag{1}$$

Therefore, the total I/O time for writing all B bundles T_{io} is represented as

$$T_{io} = max(T_1, T_2, ..., T_B) \tag{2}$$

The three procedures' performance are directly determined by the group partitioning strategy i.e 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.

The rest of this paper 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 fits 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 [8]. 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 dataset with B bundles, let the data size of bundle i be S_i, the total number of processes be n, the group size of AG_i be A_i, then A_i roughly amounts to

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

Equation 3 first substract B from n to reserve 1 process for each bundle, 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 B is larger than n, multiple small bundles can be taken as a large bundle and assigned to one process.

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.

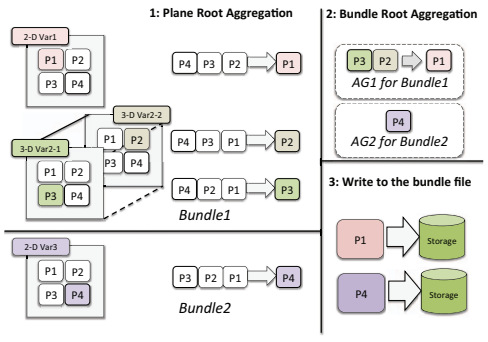

Fig. 6: Write Two Bundle Files using BPS.

Fig. 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, Var3 is a 2-D variable in Bundle2. Initially, each plane is

shared by 4 processes. Following the BPS scheme, 3 processes are assigned to AG1, which take charge of Bundle1's I/O. 1 process is assigned to AG2, which takes charge of Bundle2's I/O. Inside Bundle1, 3 processes serve as the plane root for the 3 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 who in turn receives the plane data and writes it to the storage.

The 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 the processes within each physical node may belong to two AGs, leading to reduced parallelism and locality. 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

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

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 at the end, Equation 4 makes sure that there is at least one physical node assigned to each bundle. A_i is finally set to the closest number that is a multiple of p, to make sure the smallest unit assigned to each bundle is a physical node. When n is smaller than $B \times p$, multiple small bundles can be taken as a large bundle and assigned to one physical node. 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 who actually require fewer than p processes, the overprovisioned processes should be better assigned to those larger bundles who are potential stragglers.

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_{sfl} 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 Fig. 7, Bundle1, Bundle2 and Bundle3 respectively possess 5, 3 and 1 plane(s). These planes are shared among 9 processes. Following IPS, the group sizes of AG1, AG2, AG3 are proportionally set to 5, 3, 1. Then, for each plane of Bundle1, processes P1-P5 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. Instead, if we follow BPS or LPS, every process needs to receive the plane data from all other 8 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 Fig. 7. IPS is able to reduce T_{sfl} by alleviated congestion compared to the all-to-all data shuffling in BPS and LPS.

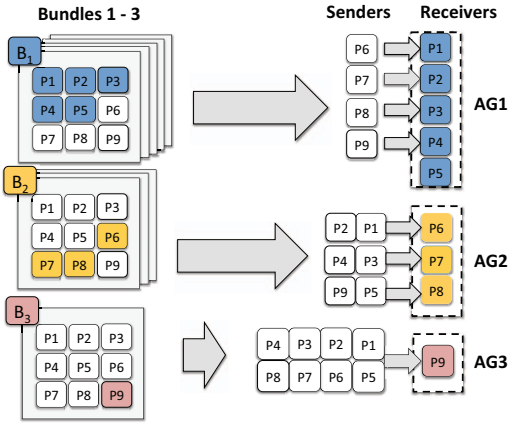

Fig. 7: Partitioning Scheme of IPS.

V. Experimental Evaluation

In this section, we systematically evaluate the effectiveness of BPAR. We compare BPAR-based GEOS-5 with original NetCDF-based GEOS-5, and analyze BPAR's three partitioning schemes using the same experimental setup as described in Section II,

A. Overall Performance of BPAR

We implement BPAR on top of GEOS-5 with BPS, IPS and LPS. Fig. 8 shows its average I/O time in each timestep as a result of the increasing number of processes.

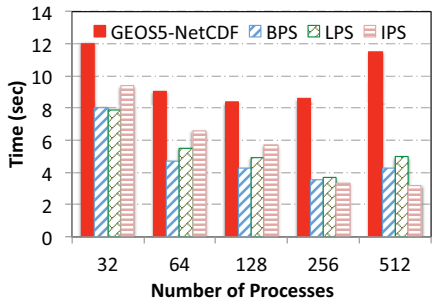

Fig. 8: Performance of GEOS-5 with BPAR.

The BPS, LPS and IPS schemes demonstrate compelling benefits over GEOS5-NetCDF. They reduce the I/O time delivered by GEOS5-NetCDF by up to 53%, 47% and 51% on average. In addition, all three cases show better scalability. This is because, unlike original 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.

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 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.

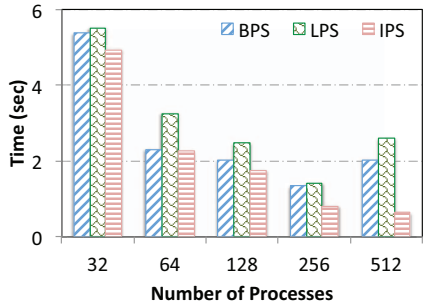

Fig. 9: The Shuffle Time of BPS, LPS, IPS.

1) Analysis of Shuffle Operation: Fig. 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 all 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 who actually starve for more resources are left unnoticed, resulting in the prolonged overall shuffle time. Table I further reveals the group size of LPS for the largest bundle 'moist' is constantly the smallest. This explains the trends in Fig. 9.

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

TABLE I: Group Size of Moist

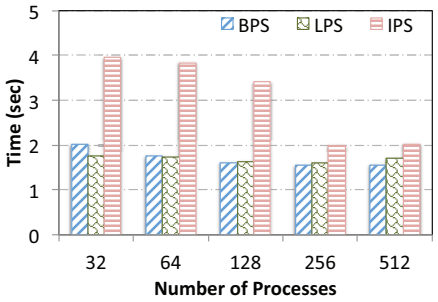

Fig. 10: The Performance of Collective Write Operation.

2) Analysis of Collective Write Operation: We have also measured the time spent on collective write operation as shown in Fig. 10. 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 8 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. This is because, in both LPS and BPS, the bundle root receives one plane from the plane root each time, the transfer unit is a plane, whose size is roughly 1MB. In contrast, the bundle root in IPS receives each plane from all its group members, the smaller transfer unit and the larger number of transfers of each process is less favored by the network. On the other hand, although LPS has better locality, the high-performance Infiniband deployed on Discover blurs such benefit, which is why LPS shows no advantage in the collective write operation.

C. Discussion

From the experimental analysis, we find GEOS-5 with BPAR delivers significantly higher throughput than the 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.

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: filesytem-level optimizations, middleware-level optimizations, and application-level optimizations.

Early work on filesystems includes the large-scale development and adoption of parallel filesystems, such as Lustre [9], GPFS [10], PVFS [11] and the effort to optimize their internal implementations, by augmenting network transfers [12], caching strategies [13], I/O scheduler [14], or through more hardware-based integrations and upgrades [15], [16], [17]. These works focus on the fundamental software layers that directly interact with the storage, which is orthogonal to our work.

The middleware-level optimizations largely center around parallel I/O techniques and I/O offloading. MPI-IO [1] is a parallel I/O middleware widely applied to scientific applications. It provides applications with parallel access to shared datasets, in addition to superior aggregation strategies to coalesce small data into larger ones. Advanced parallel I/O middleware libraries such as PnetCDF [2], HDF5 [3] are built on top of MPI-IO, while inheriting most of its features, they allow applications to access the shared dataset 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 [18], [19], [20], [21]. 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 [22].

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 [23] uses chunking to improve the data locality and the request size. Meanwhile, it alleviates the synchronization overhead by allowing users to pre-define the data format in XML file. However, this introduces extra overhead for the users, and the output in BP format is only accessible using the ADIOS interface. PLFS [8] 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 [24].

Aside from parallel I/O techniques, I/O offloading 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 [25], [26], I/O staging [27], [28] and burst buffer [29], [30]. I/O forwarding focuses on eliminating system *noise* from I/O operations by offloading I/O to dedicated I/O nodes. Most notably, it has been applied on Blue Gene/P systems. I/O staging stages the dataset 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 DRAM and SSD, and allows the actual data flushing to the filesystem to be conducted simultaneously with application's ensuing computation, thereby largely reduce the time spent on scientific applications' I/O phase.

The applicaton-level optimizations generally focus on how to port the parallel techniques to the applications for enhanced parallelism. Tian et al. [31] add ADIOS support to GEOS-5 and obtain significant performance improvement. Li et al. [32] replace the sequential I/O operation in AMR cosmology application with MPI-IO and HDF5, and point out the advantages and disadvantages of these techniques. Similarly, Johnson et al. [33] 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, we focus on researching the potential benefit that can be attained from cross-file optimizations and build the related framework to accelerate the representative application GEOS-5, together with a broad range of other scientific applications. BPAR mitigates several aforementioned deficiencies of existing parallel I/O techniques, such as the heavy I/O contention, etc.

VII. Conclusions

In this work, we have explored the major factors that restrict the I/O performance of the baseline GEOS-5, and GEOS-5 with PnetCDF. Based on our analysis, we research a Bundle-based PARallel aggregation framework named BPAR together with three of its partitioning schemes to enhance the I/O performance of GEOS-5 and a broad range of other scientific applications. The results of our experiments reveal that BPAR can achieve on average 2.1× performance improvement over the baseline GEOS-5.

In the future, we plan to research the BPAR-based partitioning schemes for the representative parallel I/O techniques such as MPI-IO, PnetCDF and build the adaptive libraries that are able to dynamically select the optimal partitioning schemes based on the scientific application workload.

Acknowledgments

This work is funded in part by a NASA grant NNX11AR20G and enabled by the U.S. National Science Foundation award CNS-1059376 to Auburn University.

References

[1] R. Thakur, W. Gropp, and E. Lusk, "Data sieving and collective I/O in ROMIO," in *Frontiers of Massively Parallel Computation, 1999. Frontiers' 99. The Seventh Symposium on the*, pp. 182–189, IEEE, 1999.

[2] J. Li, W.-k. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel NetCDF: A high-performance scientific I/O interface," in *Supercomputing, 2003 ACM/IEEE Conference*, pp. 39–39, IEEE, 2003.

[3] "HDF5 home page." http://www.hdfgroup.org/HDF5/.

[4] S. Sarawagi and M. Stonebraker, "Efficient organization of large multidimensional arrays," in *Data Engineering, 1994. Proceedings. 10th International Conference*, pp. 328–336, IEEE, 1994.

[5] Z. Liu, J. Lofstead, T. Wang, and W. Yu, "A case of system-wide power management for scientific applications," in *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, pp. 1–8, IEEE, 2013.

[6] "Discover Supercomputer Statistics." http://www.nccs.nasa.gov/discover_front.html.

[7] "NASA Center for Climate Simulation." http://www.nccs.nasa.gov/index.html.

[8] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate, "PLFS: a checkpoint filesystem for parallel applications," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, p. 21, ACM, 2009.

[9] Z. Liu, B. Wang, T. Wang, Y. Tian, C. Xu, Y. Wang, W. Yu, C. A. Cruz, S. Zhou, T. Clune, *et al.*, "Profiling and improving I/O performance of a large-scale climate scientific application," in *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*, pp. 1–7, IEEE, 2013.

[10] F. B. Schmuck and R. L. Haskin, "GPFS: A shared-disk file system for large computing clusters.," in *FAST*, vol. 2, p. 19, 2002.

[11] R. B. Ross, R. Thakur, *et al.*, "PVFS: A parallel file system for Linux clusters," in *in Proceedings of the 4th Annual Linux Showcase and Conference*, pp. 391–430, 2000.

[12] P. Carns, S. Lang, R. Ross, M. Vilayannur, J. Kunkel, and T. Ludwig, "Small-file access in parallel file systems," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1–11, IEEE, 2009.

[13] S. Byna, Y. Chen, X.-H. Sun, R. Thakur, and W. Gropp, "Parallel I/O prefetching using MPI file caching and I/O signatures," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pp. 1–12, IEEE, 2008.

[14] Y. Qian, E. Barton, T. Wang, N. Puntambekar, and A. Dilger, "A novel network request scheduler for a large scale storage system," *Computer Science-Research and Development*, vol. 23, no. 3-4, pp. 143–148, 2009.

[15] G. M. Shipman, D. A. Dillow, D. Fuller, R. Gunasekaran, J. Hill, Y. Kim, S. Oral, D. Reitz, J. Simmons, and F. Wang, "A next-generation parallel file system environment for the OLCF," in *Proceedings of the Cray User Group Conference, Stuttgart, Germany*, 2012.

[16] S. Oral, D. A. Dillow, D. Fuller, J. Hill, D. Leverman, S. S. Vazhkudai, F. Wang, Y. Kim, J. Rogers, J. Simmons, *et al.*, "Olcfs 1 tb/s, next-generation lustre file system,"

[17] S. Oral, J. Simmons, J. Hill, D. Leverman, F. Wang, M. Ezell, R. Miller, D. Fuller, R. Gunasekaran, Y. Kim, *et al.*, "Best practices and lessons learned from deploying and operating large-scale data-centric parallel file systems,"

[18] J. Liu, B. Crysler, Y. Lu, and Y. Chen, "Locality-driven high-level I/O aggregation for processing scientific datasets," in *Big Data, 2013 IEEE International Conference on*, pp. 103–111, IEEE, 2013.

[19] M. Howison, "Tuning hdf5 for lustre file systems," in *Workshop on Interfaces and Abstractions for Scientific Data Storage (IASDS10), Heraklion, Crete, Greece, September 24, 2010*, 2012.

[20] K. Gao, W.-k. Liao, A. Choudhary, R. Ross, and R. Latham, "Combining I/O operations for multiple array variables in parallel netCDF," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pp. 1–10, IEEE, 2009.

[21] C. Xu, M. G. Venkata, R. L. Graham, Y. Wang, Z. Liu, and W. Yu, "Sloavx: Scalable logarithmic alltoallv algorithm for hierarchical multi-core systems," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pp. 369–376, IEEE, 2013.

[22] W. Yu and J. Vetter, "Parcoll: Partitioned collective I/O on the Cray XT," in *Parallel Processing, 2008. ICPP'08. 37th International Conference on*, pp. 562–569, IEEE, 2008.

[23] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan, "Adaptable, metadata rich IO methods for portable high performance IO," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1–10, IEEE, 2009.

[24] W. Yu, J. S. Vetter, and H. S. Oral, "Performance characterization and optimization of parallel I/O on the Cray XT," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–11, IEEE, 2008.

[25] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan, "Scalable I/O forwarding framework for high-performance computing systems," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pp. 1–10, IEEE, 2009.

[26] V. Vishwanath, M. Hereld, K. Iskra, D. Kimpe, V. Morozov, M. E. Papka, R. Ross, and K. Yoshii, "Accelerating I/O forwarding in IBM Blue Gene/P systems," in *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, pp. 1–10, IEEE, 2010.

[27] A. Nisar, W.-k. Liao, and A. Choudhary, "Scaling parallel I/O performance through I/O delegate and caching system," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pp. 1–12, IEEE, 2008.

[28] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng, "Datastager: scalable data staging services for petascale applications," *Cluster Computing*, vol. 13, no. 3, pp. 277–290, 2010.

[29] T. Wang, H. S. Oral, Y. Wang, B. W. Settlemyer, S. Atchley, and W. Yu, "Burstmem: A high-performance burst buffer system for scientific applications," in *Big Data, 2014 IEEE International Conference on*, IEEE, 2014.

[30] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pp. 1–11, IEEE, 2012.

[31] Y. Tian, Z. Liu, S. Klasky, B. Wang, H. Abbasi, S. Zhou, N. Podhorszki, T. Clune, J. Logan, and W. Yu, "A lightweight I/O scheme to facilitate spatial and temporal queries of scientific data analytics," in *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*, pp. 1–10, IEEE, 2013.

[32] J. Li, W.-k. Liao, A. Choudhary, and V. Taylor, "I/O analysis and optimization for an amr cosmology application," in *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on*, pp. 119–126, IEEE, 2002.

[33] N. Johnson and I. Bethune, "Adding parallel I/O to PARA-BMU," *Academic Research Training and Support*, 2012.