A Lightweight I/O Scheme to Facilitate Spatial and Temporal Queries of Scientific Data Analytics

Yuan Tian[15] Zhuo Liu[1] Scott Klasky[2] Bin Wang[1] Hasan Abbasi[2]
Shujia Zhou[34] Norbert Podhorszki[2] Tom Clune[3] Jeremy Logan[5] Weikuan Yu[1]

Auburn University[1]
{*tianyua,zhuoliu,bwang,wkyu*}*@auburn.edu*

Oak Ridge National Lab[2]
{*klasky,habbasi,pnorbert*}*@ornl.gov*

Goddard Space Flight Center[3]
thomas.l.clune@nasa.gov

Northrop Grumman Corporation[4]
shujia.zhou-1@nasa.gov

University of Tennessee Knoxville[5]
{*tiany,loganjs*}*@ornl.gov*

***Abstract*—In the era of petascale computing, more scientific applications are being deployed on leadership scale computing platforms to enhance the scientific productivity. Many I/O techniques have been designed to address the growing I/O bottleneck on large-scale systems by handling massive scientific data in a holistic manner. While such techniques have been leveraged in a wide range of applications, they have not been shown as adequate for many mission critical applications, particularly in data post-processing stage. One of the examples is that some scientific applications generate datasets composed of a vast amount of small data elements that are organized along many spatial and temporal dimensions but require sophisticated data analytics on one or more dimensions. Including such dimensional knowledge into data organization can be beneficial to the efficiency of data post-processing, which is often missing from exiting I/O techniques. In this study, we propose a novel I/O scheme named STAR (Spatial and Temporal AggRegation) to enable high performance data queries for scientific analytics. STAR is able to dive into the massive data, identify the spatial and temporal relationships among data variables, and accordingly organize them into an *optimized* multi-dimensional data structure before storing to the storage. This technique not only facilitates the common access patterns of data analytics, but also further reduces the application turnaround time. In particular, STAR is able to enable efficient data queries along the time dimension, a practice common in scientific analytics but not yet supported by existing I/O techniques. In our case study with a critical climate modeling application GEOS-5, the experimental results on Jaguar supercomputer demonstrate an improvement up to 73 times for the read performance compared to the original I/O method.**

I. Introduction

High performance computing (HPC) today is seeing extremely high growth rates. From a single petaflop for RoadRunner in 2008, the Cray Titan has achieved 17 PFlops in 2012. Modeling and simulation continue to play an increasingly important role in pushing the technological envelope in almost every scientific field. Plentiful compute power in our leadership petascale machines allows us to model more complex problems, such as the impact of global warming on the planet [2], improvements to combustion technologies [6], the development of nuclear fusion [5], among other high impact sciences. The volume of generated data from the applications, and the needs of the requisite analysis demand a high performance I/O system. However, the growing gap between computing power and I/O speed has become one of the biggest challenges while HPC is evolving towards exascale.

978-1-4799-0218-7/13/$31.00

In the past few years, we have seen many scientific applications are being transmitted to large-scale systems by adopting I/O solutions that are designed to leverage the parallel storage system. Recently, emerging co-design paradigms are bringing the computer scientists and domain scientists together to design the computer hardware, software and algorithms that accommodate the computational requirements of applications. However, many applications have complex data characteristics that are not well supported by existing parallel I/O libraries. One particular challenging case is the applications that generate a large number of small variables. In parallel, each process only holds a very small portion of data for each variable. It is challenging to provide a good I/O speed for both writing and reading. Data aggregation is a common practice to consolidate the small blocks into large writes that are preferable on current storage system. Many studies [29], [15], [35] have shown the effectiveness of such strategy. However, existing aggregation techniques simply concatenate data segments without identifying the relationship among the variable data. The result is a data output that provides limited read performance, consequently degrading the efficiency of data post-processing. For example, [7] reported an overhead nearly 90% from I/O on extreme scale visualization.

The complexity of data access patterns during data post-processing further exacerbates the I/O problem. Our earlier work [31], [32] contributed to understanding the common spatial access patterns of data analytics, and proposed a data layout technique to alleviate the I/O bottleneck. However, they only examined the spatial dimension. Data analytics in temporal dimension is also commonly performed in scientific application, but poorly supported. Consider a scenario where a climate scientist attempts to observe the temperature change of a certain area during one week, which involves a spatial subset of a variable spanning multiple time steps. The data access pattern here is to read the temperature data along the

time dimension over a small spatial region. Without careful organization, a large number of seek operations are required to retrieve data in both spatial and temporal dimensions, resulting in degraded I/O performance.

To support fast queries in spatial and temporal dimensions, we have proposed a lightweight I/O scheme called STAR - Spatial and Temporal Aggregation. STAR identifies the spatial and temporal relationships between data segments, and produces a data format that can significantly improve read performance for common access patterns of data analytics [31]. Built upon our analytical understanding of the *optimized* data organization to accelerate spatial queries [32], STAR further employs a temporal aggregation as a complementary technique. Such temporal aggregation opens up another horizon for data consolidation and constructs an optimized data organization for efficient temporal data post-processing. While either aggregation technique can work well by itself, their integration into one holistic solution can help us find a balance between them, thereby enabling even higher read performance for a much larger variety of analytical and visualization operations. To the best of our knowledge, this approach has not been studied previously, but it provides an elegant mechanism for data consolidation. The novelty of the approach, as well as its potential for adoption, lies in its *improvement* in write performance in addition to its read benefits.

STAR has been incorporated into the Adaptable I/O System (ADIOS) [1], [20], a high-performance I/O middleware from Oak Ridge National Laboratory (ORNL) for HPC applications. Our initial motivating application, and thus the target of our evaluation, is the NASA GEOS-5 [2] climate simulation. We evaluate the advantages of STAR on the Jaguar [22] supercomputer at ORNL. Our results show a dramatic improvement in both write performance (up to 11x improvement) and read performance (up to 73x improvement) compared to the original GEOS-5.

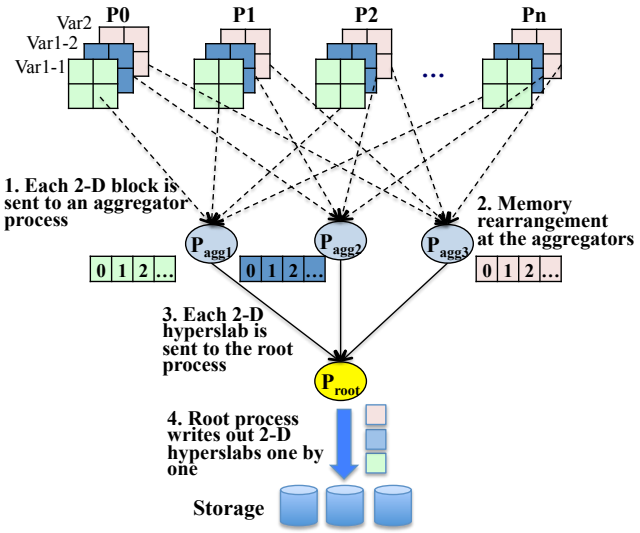
The rest of the paper is organized as follows. We first discuss the motivation of this work in detail in Section II. We then introduce the design of STAR in Section III. Section IV validates our strategy through a comprehensive set of experimental results. A literature review is provided in Section V. We conclude the paper with future research directions in Section VI.

II. Motivation

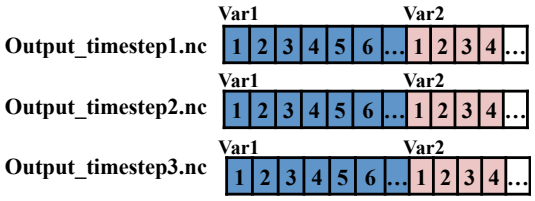
The performance of I/O on large-scale systems relies on a harmonious match between the capability of its underlying storage system, the logical data organization, and the characteristics of I/O patterns. The optimal I/O performance can be expected when all of these factors conform with each other. However, applications normally have different data organization and I/O patterns than current mainstream magnetic disk-based backend storage systems, therefore imposing substantial challenges in achieving fast I/O. In this section, we present a case study of a representative application GEOS-5, then discuss three motivating I/O issues that drive this research.

A. GEOS-5: A Case Study of Data Organization and I/O Patterns

Logically Contiguous (LC) is one of the common data organization used by many scientific applications. One of such examples is GEOS-5, the Goddard Earth Observing System Model designed by NASA to simulate climate variability on a wide range of time scales, from small scales of several hours to long-term climate changes across multiple centuries. In GEOS-5, data from each process after domain decomposition is aggregated at a few *aggregator* processes for data rearrangement before written to the storage. Such operations are required to maintain the contiguity of data, which in turn introduces overhead for data movement and memory copies. Such overhead becomes significant when a large number of variables are being written out. This pattern is also a common case for scientific applications [31].



(a) Data Movement Among n Processes in GEOS-5 for Logically Contiguous Data Organization



(b) Data Organization of 3 Time Steps

Fig. 1: Data Movement and Organization of GEOS-5

In a GEOS-5 job with n processes, the data movement to write a 3-D variable Var1 and a 2-D variable Var2 using logically contiguous data organization is shown in Fig. 1(a). A 2-D domain decomposition is performed on Var1 and Var2. As we can see, a multidimensional variable is written out in the unit of 2-D hyperslabs. One progress is designated as the aggregator for one hyperslab. After the 2-D hyperslabs are formed, one *Root* process receives them from the aggregators according to their logical position in the global array. It then

stores them to the storage. From the movement of data, we can observe a series of sequential memory and communication operations that would impact the write performance. A large number of *many to many shuffling* and *many to one fan-in* communication operations are required for data transfer, which degrade the communication and I/O performance and constrain the application scalability. Such overhead grows linearly with the increasing volume of the output variables, as well as the number of processes.

Data analytics on GEOS-5 output data also suffers from the low I/O speed. GEOS-5 generates one output file for each time step. Each variable of that time step is organized contiguously within the file, as shown in Fig. 1(b). For spatial analytics within one time step, read performance suffers from frequent seek operations when requested data subset does not match with its organization on the disk [32]. Read performance is even further degraded for temporal analytics, particularly when a subset of data is requested. Because it is not only limited by the seeks within one file, but also degraded by the overhead to operate on multiple files. Even if the variable of all the time steps are stored within one output file, many seek operations across time steps are still inevitable.

B. Research Targets

From above discussion, we identified three crucial issues that need to be addressed for this class of I/O inefficiency.

Deficiency of Current Aggregation Techniques: I/O challenge on large-scale system is particularly evident for applications with large amount of small outputs, such as GEOS-5 [2] and FLASH [16]. As small I/O requests do not conform to the characteristics of storage system, which currently is only optimized for large sequential requests. To alleviate this mismatch, it is a common practice to use some extra memory as a temporary data buffer to consolidate small data blocks into one large sequential block. However, the scope of such a buffering strategy is typically limited to a single compute node and thus not very effective when the data size per process is rather small on a single node. For example, one time step of the half degree simulation of GEOS-5 generates about 3.1GB data. When the simulation is run with 4,096 processes, each process contains only 0.78MB data for 265 variables as total. For a compute node consisting of 16 cores, one node can only accumulate a maximum of 12.5MB of data through shared memory. To further merge data, inter-node aggregation is frequently used to consolidate data across compute nodes. However, this technique involves significant network communication costs. Both effectiveness and scalability of such network dependent techniques are limited when the simulation is run at scale. An aggregation algorithm that can consolidate small data into larger blocks efficiently with negligible overhead is desired.

Deficiency in Supporting Temporal Data Analytics: The efficiency of data post-processing heavily relies on the read performance from generated simulation data. In our previous work [32], we studied how to organize multidimensional scientific data into *correct* size on large-scale storage systems to achieve the *optimized* read performance for common access

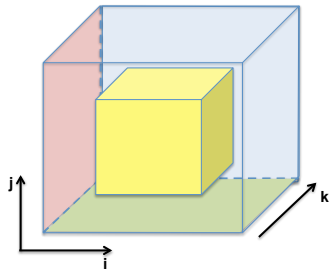


Fig. 2: A 3-D Array and Common Access Patterns(k: fastest dimension)

patterns of spatial data post-processing, including 1) reading an arbitrary full variable; 2) reading an arbitrary full subvolume; and 3) reading an arbitrary orthogonal full plane. Fig. 2 shows such patterns in a 3-D array. However, long-running applications such as GEOS-5 normally consist of many time steps. It is a common practice for scientists to explore the physical change of simulated natural systems over a good span of time. For writing, the expensive I/O costs leads to significantly prolonged simulation execution time. Simulation scientists often have to reduce the number of time steps to maintain the overall cost. Such compromise limits the scope of scientific exploration and in turn constrains the productivity. Even after the desired number of time steps are written out, the data from different time steps are stored in different files, or in different blocks within the same file. Because the data segments of different time steps are scattered, a large number of read requests have to be issued to retrieve the data, leading to degraded I/O performance. The performance is further mortified if data blocks are stored in different files as metadata overhead can become significant. This overhead grows linearly with the increasing number of time steps, making the temporal analysis of scientific data very inefficient.

Balance in Supporting Temporal and Spatial Analytics: In [32] we investigated the *optimized* data organization on parallel storage system to significantly improve read performance on a multidimensional array. However, the study only examined the physical simulation space but did not include the time axis. As both temporal and spatial analytics share equal importance in data analytics, a coordination mechanism is necessary to direct how to organize data into *correct* chunk size in order to support various data post-processing patterns in both dimensions.

To address the aforementioned I/O issues for applications such as GEOS-5 and prepare them for next-generation computing, we need to come up with a systematic I/O strategy that can further consolidate data efficiently at scale, and provide a data organization to support fast temporal and spatial data post-processing.

III. STAR: Duo-aggregation in Space and Time

To address the I/O issues described in Section II, we propose a lightweight I/O scheme that is able to identify the spatial and temporal relationships between data segments, and constructs the multidimensional data into an optimized layout

for efficient data analytics. The I/O scheme consists of two essential algorithms aiming at low-overhead data consolidation and fast post-processing: (1) Temporal Aggregation (TAR) that aggregates data chunks along the time dimension and facilitate analytics on a time series of data; (2) Spatial Aggregation (SAR) that merges data in the simulation space and speedup the analytics in the spatial dimension. Placement of the data chunks on storage system for near-optimal system concurrency. To ensure the benefits of two aggregation strategies, a coordination algorithm is also proposed to help organize data blocks into an *optimized* layout, thereby enabling efficient access for spatial and temporal analytics.

From a high level, STAR sits between the application and the underlying storage system. It uses a chunk-based data organization to enable high-performance parallel I/O flow for both writing and reading [20]. Fig. 3 shows an example of data movement using STAR to output a 2-D variable of 3 time steps. The 2-D blocks of the variable are first merged into a 3-D block with *time* as a new dimension at each process. Then SAR is performed among the processes to further merge data chunks before they are pushed to storage. Such merging is performed hierarchically to maintain the spatial locality of the data points, while reducing the amount of write requests. The placement of merged data chunk on the storage system follows the Hilbert space filling curve [14], [31]. We describe STAR in more detail in the rest of this section.

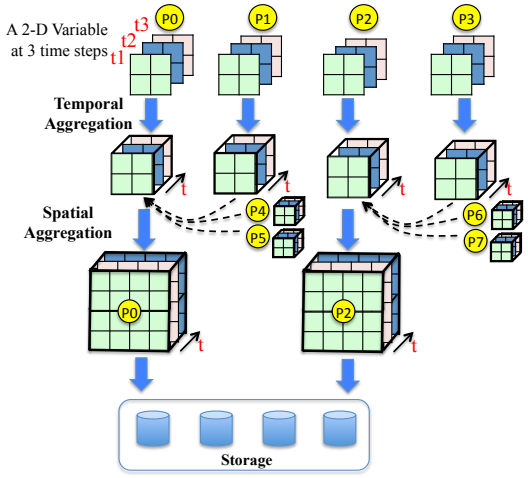


Fig. 3: Data Movement Between 8 Processes on a 2-D Variable (output from 3 timesteps)

A. Temporal Aggregation for Optimized Writing

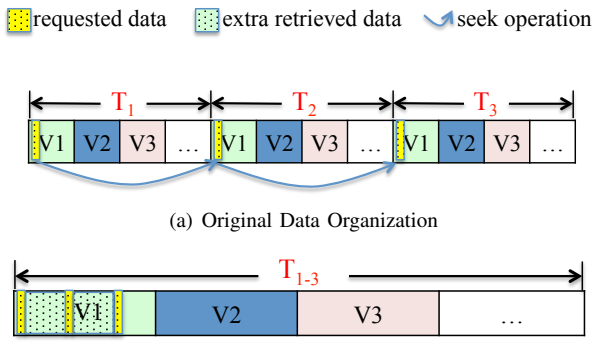
As we discussed in Section II, current aggregation techniques are limited by either the scope of the aggregation, or the expensive aggregation overhead at scale. To accommodate applications that have small output and a large number of time steps, Temporal Aggregation (TAR) is designed to further consolidate data chunks. As shown in Fig. 3, instead of pushing the variable to storage at the end of each time step, three 2-D blocks from different time steps are stored as a contiguous memory chunk within the local memory of each process. Logically three 2-D blocks are now stored as one 3-D data chunk with time as the third dimension. This strategy gives more opportunities for data to be further merged into larger segments without introducing inter-process communication overhead. In this way, a large amount of small requests are reduced to fewer large operations, which is preferable on large-scale storage systems. Better scalability can also be expected as no network communication is involved.

B. Temporal Aggregation for Efficient Temporal Analysis

The efficiency of data post-processing heavily relies on the read performance of generated simulation data. A common data layout technique is to organize the output of different time steps in different locations within one or more files. Fig. 4(a) gives an example of data output from 3 time steps into 1 file. The data from different time steps of the same variable are stored separately. Such layouts cannot efficiently support data analytics along the time dimension. To retrieve the data across multiple time steps, many small read requests are required as well as many seek operations, which are expensive on current magnetic disk-based storage system. Moreover, because the distance of data at different time steps can be large (proportional to the number of variables and the size of variables), it cannot take advantage of prefetching techniques that read additional data to avoid seeks. When a large number of time steps are requested, the cost of seeks can be significant and in turn degrades the performance. From the storage system's perspective, such organization can also be inefficient. For example, assuming that the default stripe count is 3 on a file system, the left part of Fig. 5 shows an example in which one variable is stored separately for 6 time steps. For a query that examines the data across 6 time steps, 6 requests are made to retrieve data from the storage targets. Even though such requests can be served in parallel, a large number of requests with an increasing number of time steps and readers cause contention and serialization at the storage targets. In general, the cost of such access pattern can be expressed as:

$$Cost_{org} = N_{ts} \times (T_{Request} + T_{Seek} + \frac{DataSize_{ts}}{BW_{io}}) \times \alpha \quad (1)$$

, where N_{ts} is the number of time steps of the query. $T_{Request}$ represents the overhead to initiate a read request to a storage target. T_{Seek} is the average cost of a seek operation.



(a) Original Data Organization

(b) New Data Organization with Temporal Aggregation

Fig. 4: Comparison of Data Organization (1 file)

$DataSize_{ts}$ is the requested data size per time step. BW_{io} is the I/O bandwidth of the storage target. α represents the interference factor of the system. As we can see, the cost of performing data analytics in the time dimension grows linearly with an increasing number of time steps. In particular when the requested data is small, the majority of time will be spent on initiating the I/O and performing seek operations.

By merging multiple time steps of data during writing, the output of Temporal Aggregation becomes a contiguous data segment from the merged time steps, as shown in Fig. 4(b), a more preferable data organization on large-scale storage systems. The benefit of such a data organization is that it alleviates the contention at the storage backend and improves the utilization of aggregated bandwidth. For a d_{tar} degree temporal aggregation, all of the data from N_{ts} time steps are merged. Therefore the number of I/O requests and seek operations is reduced to $\frac{N_{ts}}{d_{tar}}$. Reading from 6 time steps with the degree of TAR at 3 is shown in the right part of Fig. 5. As we can see, only two requests are required to retrieve data. The number of requests can be further reduced with higher degrees of TAR. The cost of a query in the time dimension can be expressed as:

$$Cost_{STAR} = (\frac{N_{ts}}{d_{tar}} \times T_{Request} + T_{Seek} + \frac{N_{ts} * DataSize_{ts}}{BW_{io}}) \times \alpha \quad (2)$$

Note that TAR does not impact the performance of read operations for a specific time step because the organization at each time step is not changed.

C. Spatial Aggregation with Hierarchical Topology

Based on our earlier study in [32], a Spatial Aggregation (HSA) with hierarchical topology is used to merge small data chunks in the spatial dimension. Instead of simply concatenating small chunks, HSA aggregates data chunks in a way that their spatial localities are preserved. For every spatially adjacent 2^n processes, an *Aggregation Group* (AG) is formed. Within each AG, one process is selected as the aggregator for one variable. If there is more than one variable to be aggregated, the aggregator process will be selected in a round-robin fashion within the same group for load balancing. Fig. 6 shows an example of aggregating one variable from 16 processes in a 2-D space. For every spatially adjacent 4 processes, an *AG* is formed. Aggregation is performed among the first level aggregators that hold all the data of their group

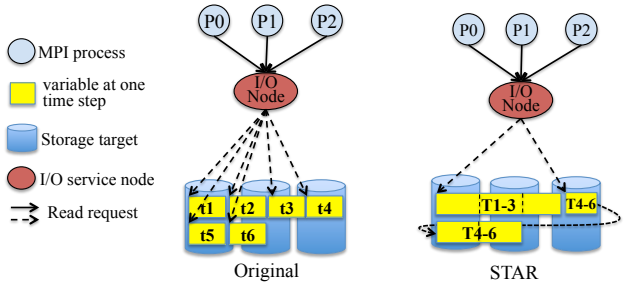


Fig. 5: Comparison of Reading on Time Domain for Different Data Organizations

members. A higher level of aggregation will be performed among the lower level aggregators. After aggregation, only the aggregators will be writing out the data. With HSA, the amount of read requests and seek operations are reduced by $level \times 2^n$ times, where $level$ is the level of HSA performed.

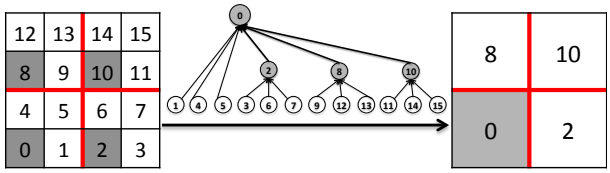


Fig. 6: Hierarchical Spatial Aggregation

D. Coordination of Duo-aggregation

In [32] we have investigated the *ideal* data organization for a multidimensional array on a specific system. For a chunk-based data organization, we defined the *optimized chunk size*, a.k.a the size of data chunk that delivers significant improved read performance for common access patterns, as $OptSize = BW_{io} \times (CC + T_s)$, where $BW_{i/o}$ is the I/O bandwidth of one storage node, T_s is the time unit for each seek operation, and CC is the communication cost unit. Such organization achieves a good balance between seek overhead and processing overhead, leading to significantly improved reading performance for common access patterns. The detail of this algorithm can be found in [32]. Given the size of an optimized chunk size $OptSize$, how much should we aggregate in the spatial dimension and how much should we aggregate in the temporal dimension? An algorithm is needed to coordinate TAR and SAR to construct small variables into the size of $OptSize$. The relationship between the two aggregation algorithms can be expressed by the following equation:

$$OptSize = d_{tar} \times DataSize_{ts} \times (2^n)^{d_{sar}}, d_{tar} \leq Total_{ts} \quad (3)$$

,where $OptSize$ is the optimized chunk size from our *Optimized Chunking* algorithm [32], d_{tar} is the degree of Temporal Aggregation, $DataSize_{ts}$ is the amount of data per time step for each process, n is the number of domain decomposition, and d_{sar} is the level of Spatial Aggregation.

As an initial study, we balance the use of TAR and SAR in the integrated scheme. By default, STAR enables one level of SAR, that is $d_{sar} = 1$. Under such circumstances, if aggregating all the time steps is to result in a chunk size larger than $OptSize$, the output will be divided into $\frac{Total_{ts}}{d_{tar}}$ times. When $OptSize$ is large enough to contain data from all the time steps, that is $Total_{ts} \leq d_{tar}$, d_{sar} can be calculated as:

$$\begin{aligned} d_{sar} &= \lfloor \log_{2^n}(\frac{OptSize}{Total_{ts} \times DataSize_{ts}}) \rfloor, \\ &= \lfloor \log_{2^n}(\frac{BW_{io} \times (CC + T_s)}{Total_{ts} \times DataSize_{ts}}) \rfloor \end{aligned} \quad (4)$$

STAR allows the user to specify the amount of memory allocated for STAR based on application and system parameters. The calculations of d_{tar} and d_{sar} are performed automatically. The coordination of TAR and SAR can be further tuned by utilizing the provenance tracking presented in [8]. For

applications that perform temporal analytics more often, more space can be allocated to the temporal aggregation and less to the spatial aggregation, and vice versa for the applications that focus more on spatial analysis.

E. STAR Implementation and Incorporation with GEOS-5

We have designed and implemented STAR first as part of the Adaptable I/O System (ADIOS)[1], which has shown significant performance benefits for a number of applications [3], [18], [20], [35], [24]. ADIOS applies the chunking strategy for storing multidimensional datasets. We built STAR as a component of ADIOS to leverage its penetration among the existing scientific applications. ADIOS allows users to specify the amount of memory for data buffering, we use such feature to direct the coordination of TAR and SAR as we described in the previous section.

As a case study, we have enabled STAR within GEOS-5 as an extension of its HISTORY I/O component, as shown in Fig. 7. HISTORY component is in charge of the output for diagnosis data. Currently it provides two types of output format: the GrDAS *flat* binary data output and self-describing NetCDF-4/HDF-5 file format. The output format of GEOS-5 is defined within its input configuration file, where users can easily switch to STAR and leverage its high-performance I/O for writing and reading.

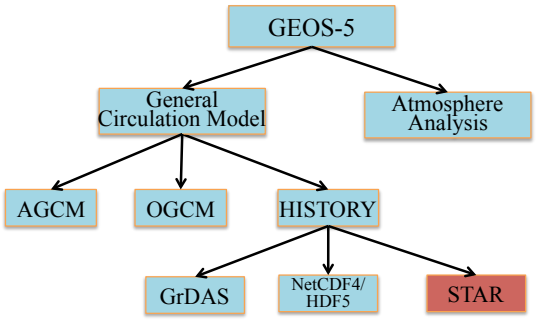


Fig. 7: GEOS-5 Model Architecture

IV. Experimental Results

We have evaluated the performance of STAR on the Jaguar [22] supercomputer at ORNL, one of the fastest supercomputers in the world. It is equipped with 18,688 compute nodes. Each node contains one 16-core Opteron processor and 32GB memory. Jaguar is connected to Spider (an installation of Lustre) as its storage subsystem. Spider has three partitions named Widow 1, Widow 2 and Widow 3, respectively. In our experiments, we use its Widow 2 partition which contains a total of 336 storage targets (OSTs).

GEOS-5 is used as the test application in our evaluation experiments. It is configured at a simulation resolution of 0.5-degree unless otherwise stated. The output consists of 185 2-D variables and 80 3-D variables in total. The simulation resolution is 576×361 with 72 vertical levels, which is regridded to 48 levels during output. Therefore the size of each 2-D variable is 576×361, and size of each 3-D variable is $576 \times 361 \times 48$. Such configuration leads to the data size per time step as 3.12GB. GEOS-5 applies 2-D domain decomposition on the simulation space, leaving the vertical resolution undivided. A 2-D variable with time as its third dimension can be expressed as var(k, j, t), where *k* represents the longitude and also is the fastest dimension. *j* represents the latitude. A 3-D variable with time as its fourth dimension can be expressed as var(k, j, i, t), where *i* represents the altitude, a.k.a., the vertical levels.

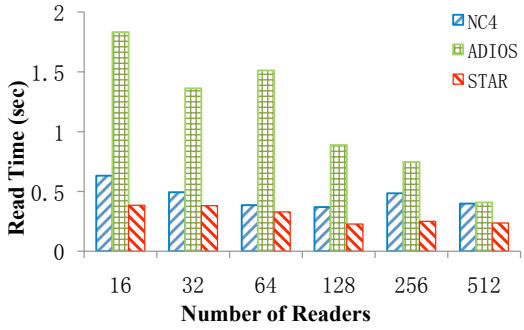
Our evaluation mainly focuses on the read performance of three different data organizations: the original GEOS-5 NetCDF-4 I/O method (NC4), the original ADIOS (ADIOS), and ADIOS with STAR (STAR). We also include the write performance evaluation to examine the performance impact of STAR to the data output. For read evaluation, the object variables are produced by 4,096 processes with 30 time steps. STAR has 2-level of spatial aggregation and 30 time steps temporal aggregation enabled during data output. Every test case is ran 10 times and the average result is reported.

A. Planar Read of 1 Time Step

As we discussed in Section II-B, reading an arbitrary orthogonal full 2-D plane from a 3-D variable is the most common and very challenging access pattern in data post-processing [31]. Therefore we mainly focused on this access pattern for read performance evaluation along with other use cases. In our first experiment, three 2-D slices, namely (k, j), (j, i) and (k, i), are read from a 3-D variable on three different dimensions. Note that temporal aggregation does not change the read performance at each time step. Therefore this experiment mainly reflects the effectiveness of spatial aggregation. To better mimic the actual practices, we vary the number of readers from 16 (the core count of one compute node on Jaguar) to 512. These choices are made based on the understanding that application scientists typically use 10% or fewer processes for reading out of the original number of writing processes. We evaluate the performance of reading three 2-D slices from a 0.5-degree variable and a 0.25-degree variable. The results are shown in Fig. 8.

As expected, intensive chunking on small variables causes the performance of ADIOSto suffer due to a large number of seek operations. Increasing the number of readers helps alleviate such overhead through parallel reads. STAR demonstrates performance improvement through its spatial aggregation algorithm. By applying SAR, the number of seek/read operations is reduced by a factor of 4 on each dimension. With 2 levels of SAR enabled, such operations are reduced to only 1/16 of the original ADIOS. Accordingly, fast reads are observed along with good scalability. A similar trend is also seen with larger variables produced by the 0.25-degree resolution simulation, as shown in Fig. 8(b). Overall, STAR demonstrates 3 times speedup compared to NC4, and 6 times speedup compared to the original ADIOS.

(a) 0.5-Degree 3-D Variable (Dimension: 576x361x48)



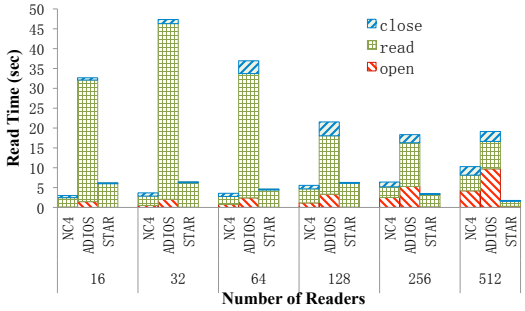
(b) 0.25-Degree 3-D Variable (Dimension: 1152x761x48)

Fig. 8: Planar Read Performance Within 1 Time Step (Total read time of 3 2-D planes on 3 dimensions)

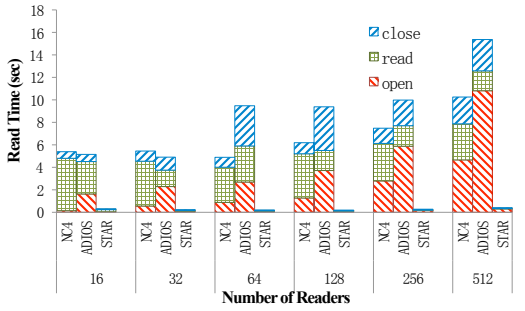
B. Planar Read of 30 Time Steps

Examining the changing values of data variables over time is a common pattern for data analytics. In this experiment, we evaluate the performance of reading a 2-D plane from 3 dimensions of a 3-D variable across all 30 time steps. That logically means that data is read in three kinds of subsets, namely (k, j, t), (k, i, t) and (j, i, t). t equals to 30 in this case. With STAR, 2-level spatial aggregation and 30-timestep temporal aggregation are performed to the data. The original GEOS-5 simulation produces one file per time step. Since temporal aggregation does not push out data immediately at the end of each time step, it reduces the number of output files by its degree of aggregation, i.e., $\frac{30}{N}$ when it is performed across N time steps. Therefore, only 1 output file is generated for STAR.

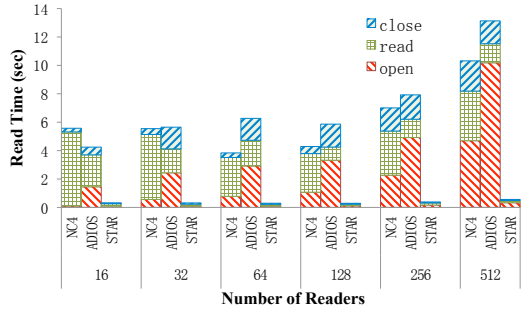
The total read time and its breakdown to retrieve a 2-D slice on three dimensions across 30 time steps together is shown in Fig. 9. Each figure represents one dimension. As we can see, the original NC4 data layout exhibits good performance in the case of (k, j, t), for which data is contiguously stored. The read time on the other two dimensions is more dominated by I/O as the data contiguity is broken on the disk. Frequent read requests along with seek operations are required to retrieve the requested data. Metadata overhead also becomes significant, particularly with larger number of readers. This is because analyzing data across 30 time steps requires each process to retrieve data from 30 files. It causes many file open and close operations that are negatively impacting the read performance. Contention in the network and at the storage targets further



(a) var(k, j, t=30)



(b) var(k, i, t=30)



(c) var(i, j, t=30)

Fig. 9: Planar Read Performance of 30 Time Steps (Multiple Files, 0.5-degree variables)

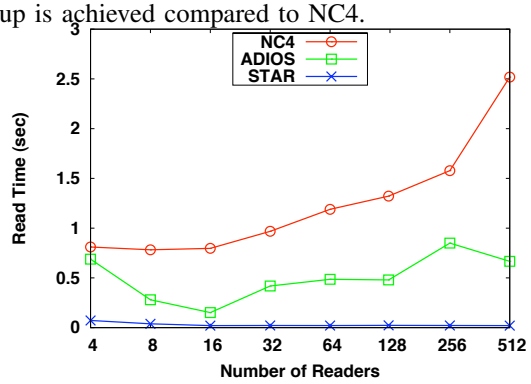
increase the cost at larger reader counts. The same metadata overhead is observed for the original ADIOS, in addition to its performance degradation from seeking through the small data chunks at each time step. STAR demonstrates significant performance benefits in this test case. Because it only generates one output file, its performance is less affected by such metadata operations. More importantly, unlike the cases of NC4 in which the majority of the time is spent on reading the data, STAR demonstrates very efficient read operations with temporal aggregation. Overall, STAR achieves the best performance on three dimensions. A maximum of 39× speedup is achieved by STAR compared to the NetCDF4 method in the original GEOS-5.

There are also cases where applications generate one single file by appending the time step outputs. In this case, the overheads of metadata operations are the same for different data organizations. Thus the performance difference mainly comes from the actual reading of data. To represent such scenario, we manually set the GEOS-5 to store data from all time steps in one file for NC4 and the original ADIOS. The performance evaluation is performed on the 0.25-degree

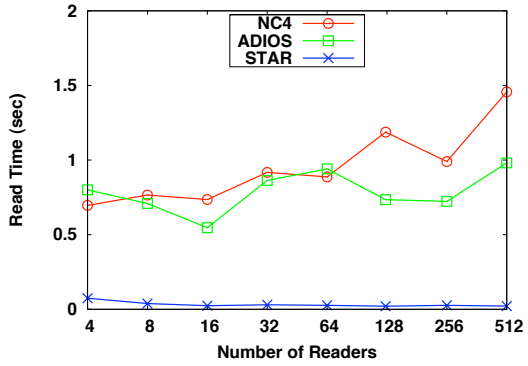
variables. Fig. 10 shows the experimental results. As we can see, STAR again achieves the best performance through its duo-aggregation algorithms . A maximum of 64× speedup is achieved compared to NC4.

C. Read Performance of 1-D Subset on 30 Time Steps

Another common data pattern for analytics is to read a 1-D subset of variables across time steps. Such access patterns can be expressed as reading a subset of the variable at (k, t), where j and i are constant, or reading a subset at (j, t), where k and i are constant. Data variables of GEOS-5 only have 48 data points in the i dimension, and reading in the (i, t) dimension is also rare in practice; therefore we do not include it in this test case. Fig. 11 shows the results for reading 1-D subsets from a 3-D variable. As we can see, while the performance of NC4 and ADIOS suffers from storage contention and a large number of read requests, STAR demonstrates a significantly improved performance in both cases. A maximum of 73× speedup is achieved compared to NC4.



(a) var(k, t)



(b) var(j, t)

Fig. 11: 1-D Read Performance of 30 Time Steps (1 file , 0.25-degree variables)

D. Write Performance of STAR with Duo-Aggregation

The aggregation algorithms potentially could negatively impact the write performance due to network communication and memory operations. However, we also expect such cost can be compensated by the intensive data buffering through temporal aggregation. So we evaluate the write performance of STAR when both TAR and SAR are enabled. In this experiment, we fix the degree of TAR to 30 time steps. Base on Equation 4, this leads to 1 level of SAR for 512 and 1,024 processes, and 2 level of SAR for the rest of test cases. We compare the write performance of STAR (duo-aggregation) with NC4, ADIOS, and STAR with only TAR enabled. The write performance is shown in Fig. 12.

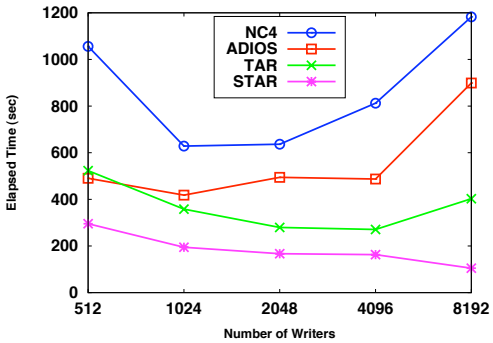


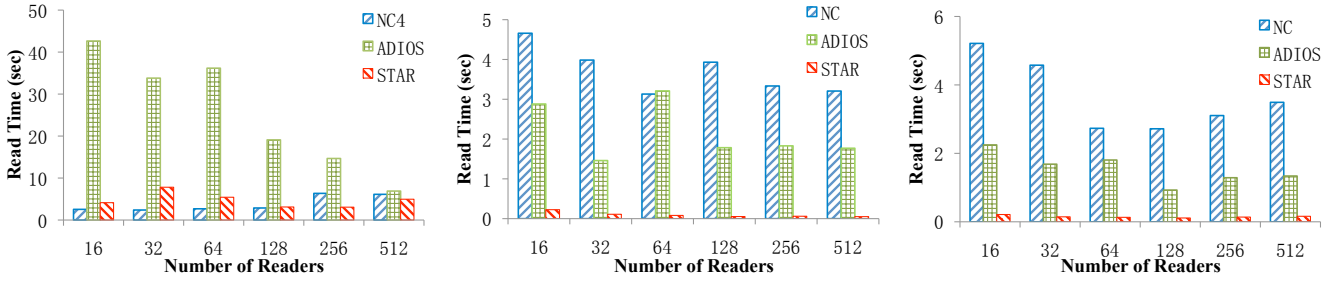
Fig. 12: Data Output Elapse Time with Duo-aggregation (30 time steps, Temporal Aggregation=30 time steps)

As shown in Fig. 12, STAR achieves the best write performance compared to NC4 and the original ADIOS. The I/O time is further reduced when both TAR and SAR are enabled. This is because SAR reduces the number of write requests by four-fold for 2-D domain decomposition, leading to further reduced contention at the storage. More importantly, good scalability is demonstrates by using such strategy. Even with 8,192 processes, we observe a performance improvement. This is because the number of write processes is reduced to 512 by using a 2-level SAR. A maximum of 11× speedup is achieved compared to NC4, and the speedup is only 4× with only TAR.

V. Related Work

Improving parallel I/O performance on large-scale system has been an active research topic in the High Performance Computing community. Early efforts such as two-phase I/O [29], split-phase collective I/O [11] and disk-directed I/O [15] tried to improve I/O performance through data buffering and scheduling techniques. Yu et al. [34] exploited the file joining on the Lustre file system and proposed a hierarchical striping strategy to fully leverage the aggregated bandwidth from storage. In [9] and [4], subfiling was also utilized to harness the I/O bandwidth. A number of I/O middleware libraries were designed to alleviate the I/O cost for large-scale scientific applications, such as NetCDF-4 [33], HDF-5 [30], [21], PnetCDF [17] and ADIOS [1]. Recently, staging [35] has been one of the major efforts to further improve I/O performance. It aggregates the data at a *staging* area, so I/O can be performed asynchronously with the progress of the application.

The performance of data read operations has gained increasing attention recently. Childs et al. [7] reported that 90% of time is spent on I/O in a data visualization workflow. To solve such issue, GFDL [12] has to store multiple copies of the data with a different dimension as the primary dimension, which requires extra I/O time and disk space. To understand the



(a) var(k, j, t=30) (b) var(k, i, t=30) (c) var(j, i, t=30)

Fig. 10: Planar Read Performance of 30 Time Steps (1 file, 0.25-degree variables)

bottlenecks of reading, Lofstead et al. evaluated and discussed the performance of many of the reading patterns for extreme scale science applications [19].

A line of work studied the efficient data reorganization for multidimensional data structure. Sarawagi et al. [25] categorized the strategies for efficient organization of large multidimensional arrays into four classes, namely chunking, reordering, redundancy, and partitioning. Chunking has been commonly recognized as an efficient data layout for multidimensional arrays because of its capability of alleviating dimension dependency [28]. [31] studied the placement of data chunks and improved read performance for scientific applications running on large scale systems. [10] and [13] examined different caching algorithm for chunking. Schlosser et al [27] explored the chunk placement strategy at the disk level. Sawires et al [26] proposed a multilevel chunking strategy to further improve the performance for range queries on a multidimensional array. Otoo et al [23] mathematically calculated the optimal size of subchunks from a combination of system parameters. One of our previous efforts [32] took a similar approach and studied the optimized chunking on large-scale systems. However, all of these works did not take into consideration of storing small multidimensional variables from a large number of processes. Nor did they examine the data organization for reading on time dimension.

VI. Conclusion

To speedup data output of the applications with large amount of small outputs and enable fast temporal and spatial data analytics, we propose a lightweight I/O scheme named STAR - Spatial and Temporal AggRegation. STAR consists of two data reorganization strategies and one coordination algorithm. Temporal Aggregation is designed to open up another dimension to further aggregate data blocks. This novel strategy also provides an efficient read performance for analytics with a temporal series of data. A spatial aggregation with a hierarchical topology is used to optimize data organization for spatial data analytics. The coordination of spatial aggregation and temporal aggregation is carefully designed to conform to our study on optimized data organization on large-scale storage systems. In addition to the reading performance benefit, the lightweight aggregation algorithms are also beneficial to the write performance through forming large data output with negligible overhead. STAR has been designed and implemented within ADIOS I/O middleware from ORNL so it can be easily adopted by any application. As a case study, we have enabled STAR for the Goddard Earth Observing System Model (GEOS-5) from NASA by extending its diagnosis data I/O component. With a much simplified I/O flow and a system-friendly data organization, an efficient I/O speed is demonstrated for both writing and reading. Our experimental results on the Jaguar supercomputer at ORNL have demonstrated a maximum of 11× speedup for the write performance, and 73× speedup for the performance of data post-processing.

VII. Acknowledgment

This work is funded in part by a NASA award NNX11AR20 and by an UT-Battelle grant (UT-B-4000103043) to Auburn University. This research used resources of the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725.

References

[1] Adaptable I/O System. http://www.nccs.gov/user-support/center-projects/adios .

[2] NASA. http://gmao.gsfc.nasa.gov/systems/geos5.

[3] H. Abbasi, G. Eisenhauer, M. Wolf, and K. Schwan. Datastager: scalable data staging services for petascale applications. In *HPDC '09*, New York, NY, USA, 2009. ACM.

[4] 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 *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, New York, NY, USA, 2009. ACM.

[5] C. S. e. a. Chang. Toward a first-principles integrated simulation of tokamak edge plasmas - art. no. 012042. *Scidac 2008: Scientific Discovery through Advanced Computing*, 125:12042–12042, 2008.

[6] J. H. Chen et al. Terascale direct numerical simulations of turbulent combustion using S3D. *Comp. Sci. & Disc.*, 2(1):015001 (31pp), 2009.

[7] H. Childs, D. Pugmire, S. Ahern, B. Whitlock, M. Howison, Prabhat, G. Weber, and E. Bethel. Extreme scaling of production visualization software on diverse architectures. *Computer Graphics and Applications, IEEE*, 30(3):22 –31, may-june 2010.

[8] J. Choi, M. Abbasi, D. Pugmire, S. Klasky, J. Qiu, and G. Fox. Mining hidden mixture context with adios-p to improve predictive pre-fetcher accuracy.

[9] A. Choudhary, W. keng Liao, K. Gao, A. Nisar, R. Ross, R. Thakur, and R. Latham. Scalable I/O and analytics. *Journal of Physics*, 180(1), 2009.

[10] P. M. Deshpande, K. Ramasamy, A. Shukla, and J. F. Naughton. Caching multidimensional queries using chunks. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, SIGMOD '98, pages 259–270, New York, NY, USA, 1998. ACM.

[11] P. M. Dickens and R. Thakur. Improving collective i/o performance using threads. In *IPPS '99/SPDP '99: Proceedings of the 13th International Symposium on Parallel Processing and the 10th Symposium on Parallel and Distributed Processing*, pages 38–45, Washington, DC, USA, 1999. IEEE Computer Society.

[12] C. H. Q. Ding and Y. He. Data organization and I/O in a parallel ocean circulation model. In *Proc. SC99*. Society Press, 1999.

[13] C. Fan, A. Gupta, and J. Liu. Latin cubes and parallel array access. In *Parallel Processing Symposium, 1994. Proceedings., Eighth International*, pages 128 –132, apr 1994.

[14] D. Hilbert. Ueber die stetige abbildung einer line auf ein flächenstück. *Math. Ann.*, 38:459–460, 1891.

[15] D. Kotz. Disk-directed i/o for mimd multiprocessors. *ACM Trans. Comput. Syst.*, 15(1):41–74, 1997.

[16] R. Latham, C. Daley, W. keng Liao, K. Gao, R. Ross, A. Dubey, and A. Choudhary. A case study for scientific i/o: improving the flash astrophysics code. *Computational Science and Discovery*, 5(1):015001, 2012.

[17] J. Li, W. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, and R. Latham. Parallel netCDF: A High Performance Scientific I/O Interface. In *Proc. SC03*, November 2003.

[18] J. Lofstead et al. Managing variability in the io performance of petascale storage system. In *Proc. SC10*, New York, NY, USA, 2010. IEEE Press.

[19] J. Lofstead, M. Polte, G. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf, and Q. Liu. Six degrees of scientific data: reading patterns for extreme scale science io. In *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, pages 49–60, New York, NY, USA, 2011. ACM.

[20] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Adaptable, metadata rich IO methods for portable high performance IO. In *IPDPS'09*, Rome, Italy, May 2009.

[21] The HDF Group. Hierarchical data format version 5, 2000–2010. http://www.hdfgroup.org/HDF5.

[22] NCCS. http://www.nccs.gov/computing-resources/jaguar/.

[23] E. J. Otoo, D. Rotem, and S. Seshadri. Optimal chunking of large multidimensional arrays for data warehousing. In *Proceedings of the ACM tenth international workshop on Data warehousing and OLAP*, DOLAP '07, pages 25–32, New York, NY, USA, 2007. ACM.

[24] M. Polte et al. ...and eat it too: High read performance in write-optimized HPC I/O middleware file formats. In *In Proceedings of Petascale Data Storage Workshop 2009 at Supercomputing 2009*, 2009.

[25] S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *Proc. 10th Int. Conf. on Data Eng.*, pages 328–336, Houston, TX, 1994.

[26] A. Sawires, N. E. Makky, and K. Ahmed. Multilevel chunking of multidimensional arrays. *Computer Systems and Applications, ACS/IEEE International Conference on*, 0:29–I, 2005.

[27] S. W. Schlosser, J. Schindler, S. Papadomanolakis, M. Shao, A. Ailamaki, C. Faloutsos, and G. R. Ganger. On multidimensional data and modern disks. In *FAST*, 2005.

[28] T. Shimada, T. Tsuji, and K. Higuchi. A storage scheme for multidimensional data alleviating dimension dependency. In *Digital Information Management, 2008. ICDIM 2008. Third International Conference on*, pages 662 –668, nov. 2008.

[29] R. Thakur and A. Choudhary. An Extended Two-Phase Method for Accessing Sections of Out-of-Core Arrays. *Scientific Programming*, 5(4):301–317, Winter 1996.

[30] The National Center for SuperComputing. HDF Home Page. http://hdf.ncsa.uiuc.com/hdf4.html.

[31] Y. Tian, S. Klasky, H. Abbasi, J. Lofstead, N. P. R. Grout, Q. Liu, Y. Wang, and W. Yu. Edo: Improving read performance for scientific applications through elastic data organization. In *CLUSTER '11: Proceedings of the 2011 IEEE International Conference on Cluster Computing*, Washington, DC, USA, 2011. IEEE Computer Society.

[32] Y. Tian, S. Klasky, W. Yu, H. Abbasi, B. Wang, N. Podhorszki, R. Grout, and M. Wolf. Smart-io: System-aware two-level data organization for efficient scientific analytics. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 181 –188, aug. 2012.

[33] Unidata. http://www.hdfgroup.org/projects/netcdf-4/.

[34] W. Yu, J. Vetter, R. Canon, and S. Jiang. Exploiting Lustre File Joining for Effective Collective I/O. In *7th Int'l Conference on Cluster Computing and Grid (CCGrid'07)*, Rio de Janeiro, Brazil, May 2007.

[35] F. Zheng et al. Predata - preparatory data analytics on peta-scale machines. In *IPDPS*, Atlanta, GA, April 2010.