CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2014; **26**:1453–1473
Published online 23 August 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3125

Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks

Qing Liu [1], Jeremy Logan [2], Yuan Tian [2], Hasan Abbasi [1], Norbert Podhorszki [1], Jong Youl Choi [1], Scott Klasky [1,*,†], Roselyne Tchoua [1], Jay Lofstead [3], Ron Oldfield [3], Manish Parashar [4], Nagiza Samatova [1,5], Karsten Schwan [6], Arie Shoshani [8], Matthew Wolf [6], Kesheng Wu [8] and Weikuan Yu [7]

[1]*Oak Ridge National Laboratory, Oak Ridge, TN, USA*
[2]*RDAV, University of Tennessee, Oak Ridge, TN, USA*
[3]*Sandia National Laboratories, Albuquerque, NM, USA*
[4]*Department of Electrical and Computer Engineering, Rutgers, The State University of New Jersey, Piscataway, NJ, USA*
[5]*North Carolina State University, Raleigh, NC, USA*
[6]*Georgia Tech, Atlanta, GA, USA*
[7]*Computer Science and Software Engineering, Auburn University, Auburn, AL, USA*
[8]*Lawrence Berkeley National Laboratory, Berkeley, CA, USA*

SUMMARY

Applications running on leadership platforms are more and more bottlenecked by storage input/output (I/O). In an effort to combat the increasing disparity between I/O throughput and compute capability, we created Adaptable IO System (ADIOS) in 2005. Focusing on putting users first with a service oriented architecture, we combined cutting edge research into new I/O techniques with a design effort to create near optimal I/O methods. As a result, ADIOS provides the highest level of synchronous I/O performance for a number of mission critical applications at various Department of Energy Leadership Computing Facilities. Meanwhile ADIOS is leading the push for next generation techniques including staging and data processing pipelines. In this paper, we describe the startling observations we have made in the last half decade of I/O research and development, and elaborate the lessons we have learned along this journey. We also detail some of the challenges that remain as we look toward the coming Exascale era. Copyright © 2013 John Wiley & Sons, Ltd.

Received 28 September 2012; Revised 18 July 2013; Accepted 19 July 2013

KEY WORDS: high performance computing; high performance I/O; I/O middleware

1. INTRODUCTION

In the past decade the High Performance Computing community has had a renewed focus on alleviating input/output (I/O) bottlenecks in scientific applications. This is due to the growing imbalance between the computational capabilities of leadership class systems as measured by floating-point operations per second compared with the maximum I/O bandwidth of these systems. In fact, although the computational capability has certainly kept up with Moore's law, the I/O capability of systems has entirely failed to keep pace with this rate of growth. Consider, for instance, the time taken to write the entire system memory to storage has increased almost fivefold from 350 s on *ASCI Purple* to 1500 s for *Jaguar* despite an increase in raw compute performance of almost three orders

*Correspondence to: Scott Klasky, Scott Klasky, Oak Ridge National Laboratory.
†E-mail: klasky@ornl.gov

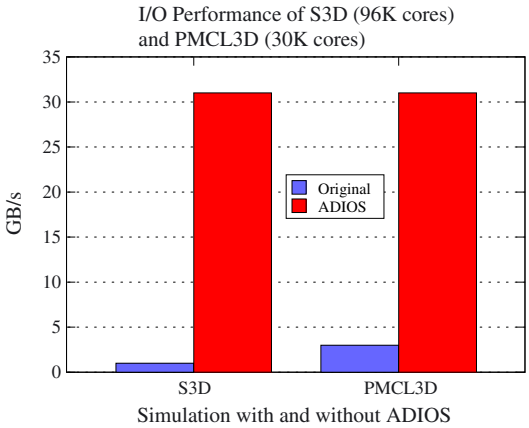


Figure 1. Adaptable I/O System (ADIOS) performance comparison.

of magnitude. In raw numbers, the I/O throughput has increased by a mere 43% from 140 GB/s to 200 GB/s. Complicating this is the fact that actually achieving this maximum bandwidth has not become any easier.

It has been clear for some time that increasingly the constraints in scaling up applications to the next order (from Terascale to Petascale to Exascale) will require a strong emphasis on data management and I/O research. In 2005, it was with this goal that the research project was started at Oak Ridge National Laboratory that lead to the development of the Adaptable I/O System (ADIOS) [1] I/O framework.

One major focus of ADIOS has been on I/O performance, where it has demonstrated superiority for many leadership applications. To illustrate this, Figure 1 compares ADIOS performance against carefully tuned MPI-IO code for two applications. As can be seen, both applications achieved approximately 30 GB/s write performance with ADIOS, as compared with less than 3 GB/s with MPI-IO.

This performance-oriented approach makes ADIOS well suited for large-scale high-performance scientific applications, such as combustion simulation (S3D), Gyrokinetic Toroidal Code (GTC), and plasma fusion simulation code (XGC). In contrast, many big data applications work with the MapReduce framework in a cloud computing environment, in which distributed data management and processing is the primary concern. Whereas the MapReduce framework seeks cost-effective computing environments by making best efforts to avoid data movement by instead moving computation to data, ADIOS pursues the most effective data management schemes to provide applications the maximum computation and resource utilization in high-performance computing (HPC) environments.

The ADIOS was developed with the understanding that we must not only address the bottlenecks for current applications and hardware platforms but also provide a path forward for the next generation of applications and systems that would need to both maximize bandwidth to the storage system and also support transparently working around the storage system bandwidth limitations with new techniques and tools. To support the diverse operating modes of both using persistent storage and other data storage and processing technology, we made a great effort to provide a simplified interface to application developers, offering a simple, portable, and scalable way for scientists to manage data that may need to be written, read or processed during simulation runs. This required abstracting away many decisions typically made in the application code so that they may be configured externally.

In addition to this focused interface with external configuration options, common services were incorporated to afford optimizations beyond those for a single platform, such as buffering, aggregation, subfiling, and chunking with options to select each based on the data distribution characteristics of the application. A variety of asynchronous I/O techniques have been investigated and are being integrated with ADIOS. A recognition of application complexity has led to new techniques for

DOI: 10.1002/cpe

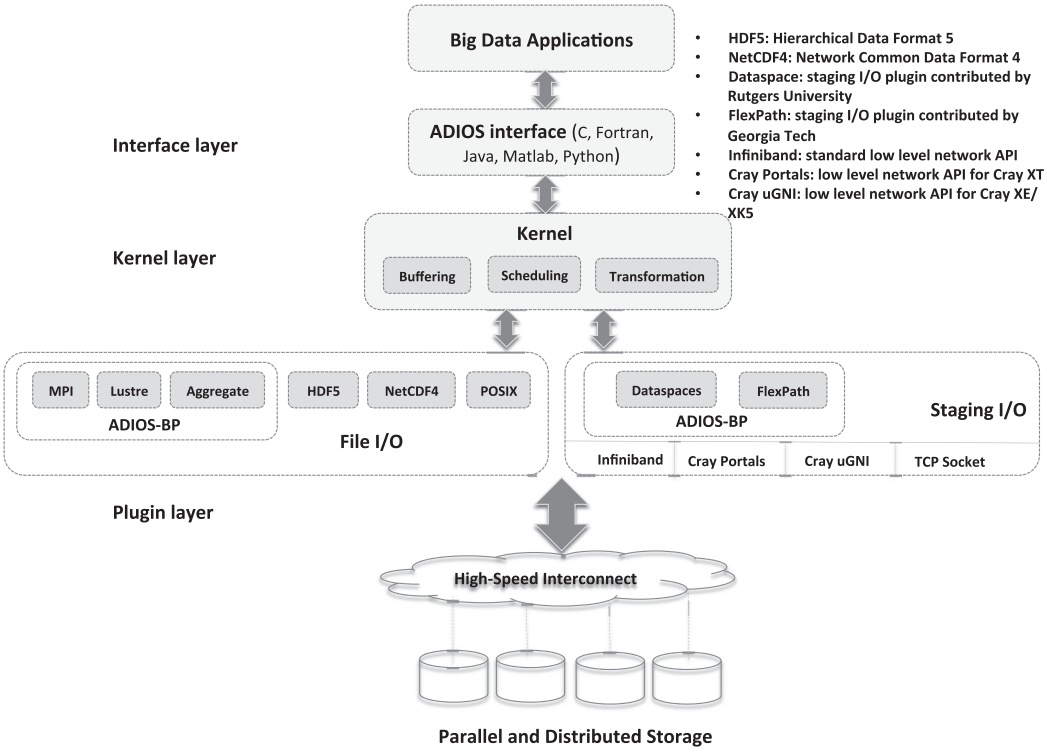


Figure 2. Adaptable IO System (ADIOS) software stack. I/O, input/output.

testing I/O performance by extracting I/O patterns from applications and automatically generating benchmark codes. And finally, the march toward Exascale has fueled the need to consider additional features such as compression and indexing to better cope with the expected tsunami of data. These features are all supported by the ADIOS software stack, which is shown in Figure 2.

2. HISTORY OF ADIOS

The increasing gap between the compute capabilities of leadership class HPC machines and their I/O capabilities has served as the motivation for a large number of research efforts into better I/O techniques [2]. This focus on improving I/O performance, in conjunction with data organization and layout that better matches the requirements of scientific users has led to the development of parallel I/O techniques such as MPI-IO [3], HDF [4, 5] and netCDF [6–8]. The evolving requirements for addressing large-scale challenges has necessitated continuous developmental efforts within these projects. ADIOS, such as these libraries, comes from a community where performance is paramount, but with a greater focus on breaking from the non-scalable paradigms of POSIX I/O semantics and file-based I/O. Parallel log-structured file system (PLFS) [9] is an ongoing effort to reinvent the file system interface for better performance in checkpoint–restart use cases. Similar to ADIOS, PLFS was designed to address the mismatch between how data is storage on parallel file systems and how it is logically represented within the application. The PLFS approach uses a user-level file system interface to provide some of the performance benefits of ADIOS without requiring substantial changes in the application source code. Recent work on PLFS [10] also leverages the enhanced metadata available to HDF5 to provide a semantic aware data storage layer while maintaining its performance advantages. This alternate approach maintains the traditional semantics for I/O, but is limited to moving data only to storage without leveraging *in situ* computation to reduce the data management burden on applications.

DOI: 10.1002/cpe

The initial impetus for the development of ADIOS was from the challenge of data management for the visualization of fusion simulations such as the GTC [11]. In particular, the problem that fusion scientists faced was the difficulty in running large-scale simulations that produced vast volumes of data. This data was required not just for checkpoint–restart, but also to provide scientific insights through analysis and visualization. The fusion scientists were not interested in spending a significant development effort in optimizing the application's I/O for a single platform, only to find the next iteration of the supercomputer make moot their hard work. Moreover, the best performing I/O technique at the time, single file per process, had started to become bottlenecked by the metadata operations. As applications scaled, the number of files was also becoming unmanageable. The alternate solution of creating a logically contiguous (LC) file using MPI-IO also suffered from limited scalability. The additional synchronization and communication required to organize the data in the file was a scalability bottleneck, limiting applications such as S3D [12] and GTC from scaling to the largest supercomputers.

Two key insights have driven the design of ADIOS. First, users do not need to be aware of the low-level layout and organization of scientific data. Thus, a smart I/O middleware can organize the *bits* in any manner, so long as the actual information in the data is maintained. Second, scientific users should not be burdened with optimizations for each platform (and for each evolution of that platform) they use. Thus, any I/O middleware seeking to solve the data challenges should be rich enough to provide platform specific optimizations for I/O without increasing the complexity of the I/O routines.

Both insights can be explained with a simple example. Consider a simulation that produces data from each parallel process, to be later consumed as a global array for visualization. It is of no significance if the processes output data into separate files or into a single file, as long as the data can be read using global offsets. Moreover, optimizing the data output procedures for a specific configuration of the parallel file system is not important to the user. Parameters such as stripe size, stripe count, and so on do not add any meaningful semantics to the data itself and should not be part of the application itself.

A secondary, though just as significant insight motivated the design of ADIOS. The ever increasing mismatch between the scalability of I/O and computation creates an inherent scalability limit for synchronous I/O. Asynchronous file system efforts such as the light weight file system (LWFS) [13] have been presented as an alternate to both the metadata and synchronous I/O obstacles. The tight security and isolation requirements for supercomputing centers, however, affected the impact that such efforts could have on real applications. Without a proper security and code review, no supercomputing center would allow a new research file system to be deployed on production machines.

Based on these observations ADIOS was born with three key ideas.

1. *Abstraction of the I/O technique*. As noted earlier, the burden on the computational scientists was increasingly become unsustainable. ADIOS introduced a simple abstract API for the developer to output data from her application, while allowing a user to select a specific I/O method that was optimized for the current platform.
2. *Support for data processing pipelines.* Instead of pushing more functionality into the file system, or even radically modifying the file system, ADIOS allowed users to define additional *staging nodes* that could easily replicate the functionality found in file systems such as LWFS. Utilizing asynchronous buffered data movement techniques minimized the time the application spent waiting on I/O resources. The use of fully functional computational nodes for staging also opened up new avenues for *in situ* data processing, for visualization, analysis or even data reorganization.
3. *Service oriented architecture*. The most fundamental idea in ADIOS was to provide a consistent interface to the application developer, while still allowing developers to create new I/O techniques that could address their specific challenges. By enabling these services, ADIOS provided a research platform for the development of new techniques in high performance I/O. This, in turn, also allowed new I/O methods to be easily evaluated and tested with new applications and on new platforms, and greatly enriched the choices available to the users.

DOI: 10.1002/cpe

3. ADIOS DESIGN

Application developers had been using various I/O solutions, such as FORTRAN's native I/O, MPI-IO, HDF5 or ad hoc solutions to manage the scientific data. Given the sometimes vast differences between each new HPC platform deployed, these techniques did not retain their performance as the code was moved to these new systems. The I/O strategy had to be redesigned, and the code rewritten for each new platform. Although developers are usually enthusiastic about writing new parallel computation code for new programming paradigms and new architectures, the I/O routines are frequently perceived as only a burden causing little effort to be expended on them. Moreover, the complexity of the I/O subsystem is not documented well requiring experimentation to figure out what works well on a given system for a given application size and data distribution. As a result, scientists avoid reworking I/O routines and instead regularly scale down the output size to the bare minimum to avoid spending too much of the allocated computer time waiting for I/O to complete. Some applications even skip writing checkpoint restart files when running on hundreds of thousands of cores risking wasting valuable allocation time if any fault causes the application to abort prematurely.

The lesson learned is to separate the listing of I/O operations in the application code from the I/O strategy employed on the current platform and for a given run size. The simplified ADIOS API avoids hiding complexity and performance problems in the IO routine calls by eliminating the variety of options available in the application code. What this yields is a simple description of the variables to be written in the application code with an external configuration to declare what to do with the data. It could be written to storage or passed to some in-flight data processing framework with no knowledge of the host application code. By separating these concerns, erroneous code for the I/O routines due to the misunderstandings and incorrect assumptions by a developer from studying the documentation and the examples is avoided. It also affords incorporating new data management techniques that may not have been envisioned when the I/O routines were initially developed.

The basic design decisions for ADIOS have been the following:

- Simple programming API that does not express I/O strategy, but instead just declares what to output,
- XML-based external description of output data and selection of I/O strategy (including the selection of multiple techniques for a single output operation),
- Multiple transport methods selectable at runtime,
- Self describing, log-based file format combined with buffered writing for best possible write performance.

At a high level, ADIOS is an I/O library that consists of write, read API along with a few utilities, for example, to query and convert the data. The library itself involves very minor overhead and has only about 40 API's for C bindings.

3.1. Simple API

The write API of ADIOS is designed to be as close to the POSIX API as possible. The single necessary extension was the adios_group_size call as a way to more easily support effective buffering and efficient file layout for maximum performance. Initialization and finalization calls in the ADIOS framework affords opportunity for each transport method to perform operations such as connecting to external resources, pre-allocating resources, and ultimately cleaning up these resources during the finalize call. The output should be 'open'ed (see Listing 1), and the size of the data of the given process is going to write in total is provided. Simple write statements are used per variable and a close statement ends the list of I/O statements signaling the end of the I/O operation.

The rest of the definition is given externally in an XML file, for example, defining that 't' is a double array of 'NX' and 'NY' scalar variables in the code, and it should be known as 'temperature' for readers of the output file. The semantics of the API only declares that any variables listed in the

DOI: 10.1002/cpe

```
adios_open (&fd, "analysis", filename, "w", &comm);
adios_group_size (fd, groupsize, total);
adios_write (fd, "NX", &NX);
adios_write (fd, "NY", &NY);
adios_write (fd, "temperature", t);
adios_close (fd);
```

adios_write statements are safely copied, and likely written, when the adios_close() returns. Assuming the output is destined for disk, the actual file-open operations may occur later then adios_open() call and writes happen usually during the adios_close() call. In the case of asynchronous I/O, the write operations will likely take place after the adios_close() call completes.

The simple semantics of the ADIOS API allows specific methods to optimize their behavior for performance or functionality without modifying the semantics and breaking the portability of the code. Despite the simplicity of the ADIOS API, it should be emphasized that the self-describing nature of ADIOS, as well as the lack of any byte level storage specification, gives ADIOS more freedom for optimization compared with lower level I/O APIs such as POSIX or MPI-IO whose APIs stipulate where each byte should be placed. In contrast, ADIOS methods are free to arrange data in whatever manner provides optimal performance, power, or resilience. Direct performance comparisons of ADIOS to MPI-IO or POSIX are thus more difficult to make.

On the read side, ADIOS read API provides a uniform interface for handling both files and data streams. The design of read API allows a user code to be oblivious of the data being handled, regardless whether it's a stream or file. It supports a rich set of data selections, including bounding box, points and data chunks, and allows a user code to advance to a certain step of the data and perform read operations, which fits nicely to the majority of scientific applications that solves problems iteratively.

3.2. XML-based external description

ADIOS uses an external XML file to describe I/O characteristics such as data types, sizes and select I/O operations for each I/O grouping. As such, the I/O routines in the user code can be simplified and transparently change the way data is processed. In particular, the XML includes data hierarchy, data type specifications, grouping and which method(s) to use to process the data. The application calls for outputting the data can be generated automatically from the XML file and included at the right place in the application code using preprocessor directives. This simplifies any changes necessary should the output need to change. Simply change the XML file, regenerate the included API calls and recompile. The only requirement is that the variables referenced in the XML file should be visible at that location in the code. All metadata information, including the path of a variable and any attributes providing extra information can be added and modified later in the XML file without recompiling the code. The same applies for selecting the actual method(s) to use for a particular run. This separation of the definition and organization of data in the self-describing, metadata rich output affords defining a generic schema for automatic visualization purposes [14] and for generating I/O skeleton applications representing large-scale codes (Section 4.6). Recent changes have introduced the possibility of encoding all of the information contained in the XML file into the source code to avoid the need for another file during execution, but it is not recommended due to the lack of flexibility this imposes. In spite of this limitation, this option has proven popular with a small subset of users.

3.3. Transport methods

The ADIOS provides methods for three process-file patterns: (i) N-to-1, that is, single file written collectively by all processes, (ii) N-to-N, that is, each process writing to a separate file, and (iii) N-to-M, that is, grouping processes to a limited number of output files, by aggregation. N-to-1

DOI: 10.1002/cpe

methods include the Message Passing Interface ('MPI') method that uses MPI-IO calls, but due to the local buffering and the output file format, cross-communication among processes for data reorganization is avoided achieving the best possible performance for MPI-IO. Other N-to-1 methods write popular file formats, such as parallel HDF5 and NetCDF4. These transport methods are there for user convenience rather than for ultimate performance because these methods are inherently limited by the performance of the underlying HDF5 and NetCDF4 libraries. The 'POSIX' transport method is an N-to-N method, where each process' buffered data is written using a single call each into a separate file. It simply avoids the optimizations provided and defenses developed by parallel file systems for complicated I/O patterns and directly uses their basic functionality and bursts data with high bandwidth. It also writes a metadata file that affords users dealing with the collection of files by a reading application as a single file. The N-to-M method aggregates data to a subset of processors and then, such as the N-to-N method, writes separate files. Currently, this method is the fastest and most scalable ADIOS transport method. All applications using ADIOS apply this method when running with more than 30,000 cores.

3.4. ADIOS-BP file format

The file format designed for ADIOS provides a self-describing data format, a log-based data organization and redundant metadata for resiliency and performance. Self-describing formats such as HDF5 and NetCDF are popular because the content of a file can be discovered by people long after the developers and their independent notes on the content are gone.

The log-based data organization affords ADIOS writing each process' output data into a separate chunk of the file concurrently. In contrast with logically contiguous file formats where the data in the memory of the processes has to be reorganized to be stored on disk according to the global, logical organization, this format eliminates (i) communication among processes when writing to reorder data and (ii) seeking to multiple offsets in the file by a process to write data interleaved with what is written by other processes. Coupled with buffering by the processes, discussed in the next section, that exploits the best available I/O bandwidth by streaming large, contiguous chunks to disk, the destination format itself avoids bottlenecks that would hamper that performance. The many processes writing to different offsets in a file or to different files even are avoiding each other on a parallel file system to the extent possible. In most cases, each process attempts to write to a single stripe target to avoid the metadata server overhead of spanning storage targets. The reading performance of this choice was shown to be generally advantageous as well [15].

4. LESSONS

In the succeeding discussions are the lessons learned and knowledge gained through working closely with users on parallel I/O. We believe these experiences on leadership computers are not only beneficial to ADIOS users but also relevant to the entire HPC community as we move forward into Exascale era.

4.1. Buffering and aggregation

Buffering and aggregation are important techniques to improve storage I/O performance particularly for large-scale simulations. These techniques effectively reduced unnecessary disk seeks caused by multiple small writes, and make large streaming writes possible. Nevertheless, they need to be carried out carefully to ensure scalability and reduce contention.

Since the ADIOS 1.2 release, a new I/O method, MPI_AMR, incorporates multilevel buffering and significantly boost I/O performance even for codes that write/read only a tiny amount of data. In this method, there are two levels of data aggregation underneath the ADIOS write calls. This is intended to make the *final* data chunks as large as possible when flushed to disk, and as a result, expensive disk seeks can be avoided. At the first level, data are aggregated in memory within a single processes for all variables output by the adios_write statements, that is, a write-behind strategy.

DOI: 10.1002/cpe

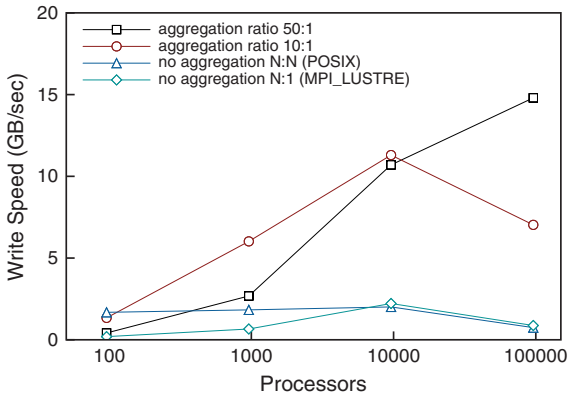


Figure 3. Aggregation versus no aggregation for a combustion simulation.

For the example in the ADIOS code earlier, the variables *NX, NY* and *temperature* in the adios_write statements will be copied to the ADIOS internal buffer, the maximum size of which can be configured through ADIOS XML file, instead of being flushed out to disk during each adios_write call.

Meanwhile, a second-level of aggregation occurs between a subset of the processes (Figure 3). This builds the buffers larger by combining the relatively small amount of data each processes has to output (after the first-level aggregation). A good example is the S3D combustion code [16]. In a typical 96,000-core S3D run on JaguarPF, each processor outputs less than 2 MB total. In this case, many small writes to disk hurt I/O performance. As has been shown elsewhere, making data larger using MPI collectives to exchange bulk data between processors can be costly [17, 18]. Here, we argue four reasons that aggregation technique, as implemented by ADIOS, can be beneficial.

1. The interconnects in high-end computing systems are becoming faster and faster. For example, the Cray Gemini interconnect on Cray XK6 can sustain up to 20 GB/sec [19]
2. The issue with collective operations in MPI-IO is not the volume of data to exchange. Instead, the dominating factor that slows down application performance is the frequency of collective operations and possibility of lock contention. As discussed in Section 3.1, the design of the MPI-IO API also makes optimization of these codes more difficult. Our earlier work [20] shows that MPI_Bcast is called 314,800 times in the Chimera run, which take 25% of the wall clock time.
3. The collective operation in ADIOS is carried out in a very controlled manner limiting internode communication. All MPI processes are split into subgroups, and aggregation is carried out within a subcommunicator for a subset of nodes reducing contention between groups. Meanwhile, indices are generated first within a group and then sent by all the aggregating processes to root process (e.g., rank 0) to avoid global collectives. This is similar to the approach taken in the ParColl [17] paper.
4. Most of today's computing resources, such as the Jaguar Cray XK6, use multicore CPUs; thus, aggregation among the cores within a single chip is inexpensive as the cost is close to that of a *memcpy()* operation.

4.2. Subfiles

Subfiles offer several key advantages over *one file* and *one file per process*. The latter two usually yield similar performance as subfiles at small core count. However, at large scale, they pose heavy pressure to either storage target or metadata servers, and therefore are not feasible.

One file requires all processes to collectively negotiate the proper write offsets. This usually involves two rounds of MPI collective calls among the processors that participates in I/O, that is, MPI_Gather to collect local sizes and an MPI_Scatter to distribute the offsets to individual

DOI: 10.1002/cpe

processes, which can be very costly *at scale*. Splitting communicators into smaller groups with each group writing to a subfile can work around this issue. However, writing to one shared file can cause serious lock contention, particularly at scale, which essentially serializes I/O from individual processes and slows down the I/O speed. One way to tackle this problem is to make the write block-aligned (i.e., stripe aligned for Lustre). However, if the size of data from each process varies, it is hard to pick the right block/stripe size that makes the access aligned without incorporating some amount of empty space. Additionally, parallel file systems such as Lustre often provide default file striping designed to limit I/O pressure from other users. The downside to this artificial limitation is the strict reduction in the maximum parallel I/O performance for this file. Subfiles are a natural work-around solution that can fully utilizes storage resources, albeit selfishly for a single application.

One file per process is a special case of subfiles where the number of subfiles equals the number of processes. However, one file per process is not scalable due to poor metadata performance for creating files for every process and therefore, more sophisticated management is needed.

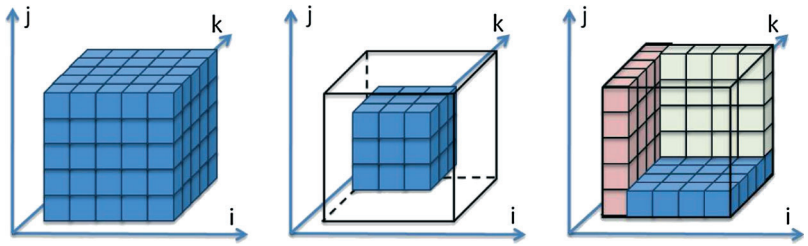
Subfiling is a compromise between one file and one file per process and makes writing significantly faster. The design objectives of subfiling are as follows: (i) introduce no additional complexities from a user perspective. Writing and reading subfiles should be as simple as writing and reading from a single, shared file; (ii) offer significantly higher I/O bandwidth at scale than the other process to file decompositions.

4.3. Balancing read and write performance

Many efforts, both past and present, have focused heavily on improving the I/O performance by studying the output side of the problem, but the read performance of scientific applications on large-scale systems has not received the same level of attention, despite its importance to drive scientific insight through scientific simulation, analysis workflows, and visualization. Worse yet, current write techniques often overlook the read side of the I/O equation and, as a result, have a substantial negative impact on read performance. Our strategy with ADIOS is to provide a favorable data layout through data reorganization during output, without introducing significant write overhead.

To improve read performance, a thorough understanding of application's access patterns is crucial. Based on the authors' direct experience with many application teams in the USA and beyond, including combustion (S3D [16]), fusion (GTC [11], GTS [21], XGC-1 [22]), earthquake simulation (SCEC [23]), MHD (Pixie3D [24]), numerical relativity codes (PAMR [25]), and supernova (Chimera [26]) codes, there are four main fundamental reading patterns for application data analysis:

- Read all of a single variable (c.f. Figure 4(a)). This would be a representative of reading the temperature across a simulation space, for example.
- Read an arbitrary orthogonal subvolume (c.f. Figure 4(b)).
- Read an arbitrary orthogonal full plane (c.f. Figure 4(c)).
- Read multiple variables together. This would be a representative of reading the components of a magnetic field vector, for example.



(a) Variable (b) Subvolume (c) 2-DPlanes

Figure 4. A $5 \times 5 \times 5$ array (k: fastest dimension). (a) Variable, (b) subvolume, and (c) two-dimensional (2D) planes.

DOI: 10.1002/cpe

Other reading patterns are either composed of a mixture of these patterns or are minor variations. For example, reading entire checkpoint–restart datasets can be perceived as an extended case of reading multiple variables together. Among these patterns, reading orthogonal planes has been the least studied. However, it is a very commonly used reading pattern by scientific applications. For example, for combustion studies with S3D [27], the computation was targeted at a variable of $1408 \times 1080 \times 1100$ points (12GB), but the majority of analysis is performed on the orthogonal planes of the variable (either 1408×1080 or 1080×1100 points). However, the performance of reading such planes is often bottlenecked by the extremely poor performance when retrieving data along slow dimensions from multidimensional arrays [28].

There are two main issues faced by such access patterns of multidimensional arrays. The first is the discrepancy between the physical limitations of magnetic storage and the common access patterns of scientific applications. Physical disks in most HPC systems are optimized to access one-dimensional large blocks of sequential data, whereas scientific data is normally multidimensional. Mapping from a n-dimensional logical space of the scientific data to a one- dimensional space of storage results in the non-contiguous placement of data points on non-primary dimension. This causes read performance to suffer due to either significant disk operation overhead such as excessive seek time, or data overhead caused by reading additional data to avoid seeks for many popular data organization strategies. A current popular solution to this problem is to store multiple copies of the same data with a different dimension being used as the primary dimension in each copy. For example, climate researchers at the Geophysical Fluid Dynamics Laboratory make multiple replicas of all datasets with x, y, z and *time* as the fastest dimension, respectively. Such work-arounds help reduce the reading time [29], but increase the total storage size by four times. data on fast dimension or slow dimension is stored on one storage target.

Second, the peak aggregated bandwidth of parallel storage systems cannot be effectively utilized when only a subset of the data is requested. This occurs because the requested data is concentrated on a very small number of storage targets with current data placement strategies, causing a substantial performance degradation and limited scalability. For example, the two-dimensional plane on the fast dimension of a three-dimensional array with LC data organization can be easily stored on one or very few storage nodes. Systems such as the Gordon [30] supercomputer at San Diego Supercomputer Center attempt to address this problem through new hardware such as solid-state devices (SSDs). However, without optimizing data organizations, even SSDs cannot maximize concurrency and therefore, peak bandwidth. Whereas disk seeks are no longer an issue with solid state storage, aggregate performance achievable based on data distribution is still an issue.

Currently there are two popular data organizations: LC and chunking. Figure 5 compares these two data organizations and show how the read performance can be different between these organizations. In the figure, a two-dimensional array with 9×9 integer elements is written on three storage targets using LC and chunking, respectively. The stripe width is equal to 36 bytes. The arrowed lines represent the order in which these data elements are stored on storage devices, for example, Object

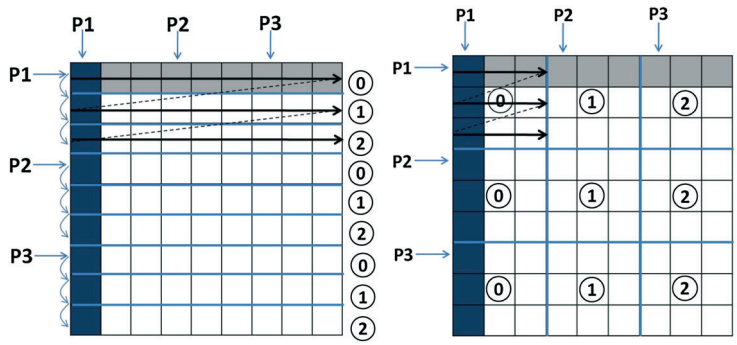


Figure 5. Current data organizations. (a) Logically contiguous and (b) chunking.

DOI: 10.1002/cpe

Storage Targets (OSTs) in the case of Lustre file system. The circled numbers indicate targets on which data elements are located; the shaded squares are the requested data elements. If we read in the row-major order with three processes, for both organizations, each process needs one seek operation and one read operation to retrieve the data. However, chunking is expected to be three times faster than LC because LC serializes read requests from three processes to one OST. Data concentration issue is observed for both LC and chunking, where either

A line of work has proven that overall *chunking* outperforms *LC* as a data organization for multidimensional data, as it is able to alleviate *dimension dependency* [31], that is, the performance of a query is not dependent on the size of the query, but the dimension. ADIOS employs chunking data layout for multidimensional arrays. However, such a strategy still faces the concurrency issue shown in the previous example. Simply placing data chunks in a round-robin fashion on storage targets does not guarantee the maximum aggregated bandwidth for all of the common access patterns, potentially limiting the read performance to the bandwidth of very few storage nodes. To address the aforementioned issues, we have designed a data placement strategy that utilizes a space-filling curve [32] mathematical model to reorganize scientific multidimensional datasets. We chose the Hilbert curve [33] in particular to leverage its unique properties for clustering and declustering capabilities as compared with the other types of space-filling curves [34]. Instead of placing data chunks in the round-robin fashion, such a strategy reorders the data chunks along the Hilbert curve ordering before they are sent to storage. By using this strategy, data from scientific multidimensional arrays can be distributed in a balanced manner across all storage devices so that the aggregated bandwidth can be effectively aggregated and exploited for challenging read access patterns, particularly planar reads. Figure 6 gives an example where data concurrency is improved three times on the slow dimension by using a Hilbert space-filling curve [33] based placement order compared with the original round-robin placement strategy. The overall data organization can be either single-file output or multiple subfile output that ADIOS supports.

4.4. I/O variability

Many current HPC systems have excellent nominal I/O throughput. Consistently achieving these levels of throughput relies on the assumption that no one else is accessing the file system while the I/O operation is performed. Our observations with many production codes show that I/O speed varies significantly from run to run due to interference from other users. As a result, the average I/O rate is significantly lower and the variance in total I/O time makes it hard to predict job wall time. The root cause of such a huge variability is that the file system level provides no quality

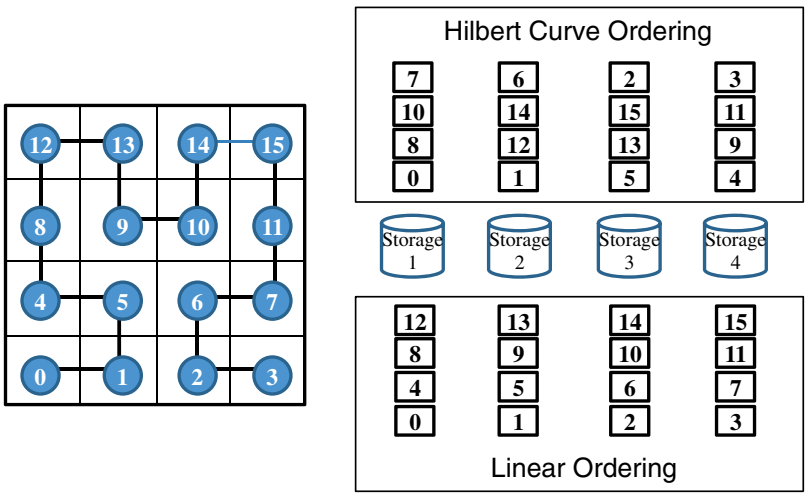


Figure 6. Comparison of linear placement and Hilbert curve placement order of 16 chunks on four storage nodes.

DOI: 10.1002/cpe

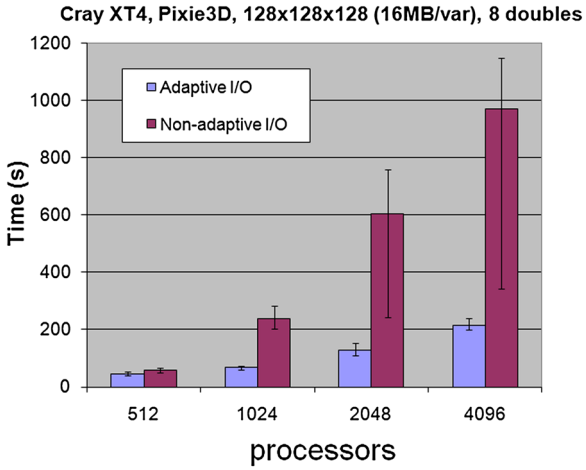


Figure 7. Adaptive input/output (I/O) versus non-adaptive.

of service, and resources such as storage targets and network interconnections are essentially shared by multiple users in a 'best-effort' manner. Simultaneous I/O operations can interfere with each, causing the observed penalties.

We attempt to resolve this problem by providing an adaptive control inside ADIOS. First, ADIOS uses subfiling to split writing so that writes can happen independently instead of collectively. By tracking the status of each storage target, a process can make the decision to switch from a slow storage target to a faster one on the fly. In particular, we have implemented an I/O control plane through which storage target state is periodically exchanged among all processes. The key advantages of adaptive I/O are twofold. First, the scheme improves overall I/O performance by actively avoiding congestion hence reducing the overall I/O variability and load imbalance, which is beneficial to both users and the system as a whole by reducing the amount of time storage targets are occupied by an application performing I/O. Second, the control framework is very lightweight and adds no additional complexities to the file system layer. Experimental results shown in Figure 7 are clear: adaptive I/O can effectively improve the write performance by a factor of 500% at 4096 processes. More importantly, the variability indicated by the error bars is much smaller in adaptive I/O.

4.5. Asynchronous data movement

Beyond buffered output, as described in Section 4.1, the next step to further reduce the impact of I/O on application performance is to introduce asynchronous I/O. One advantage of developing an I/O framework targeting modern architectures is the availability of innovative low-impact remote memory access methods for the Cray SeaStar2+ and commodity Infiniband networks. Our experience relying on the file system to provide asynchronous communication has been less than satisfying [35]. Due to both the complexity of file systems and the difficulty in adding new features, it was not viable to add asynchronous functionality to center wide file systems. Replacing the file system with a system such as LWFS [13] was not an option either due to the lack of maturity as a production tool and the reluctance by the machine administrators to introduce an unproven tool into a key portion of the production environment. Instead, extend the idea of a 'data buffer' to include a remote memory location to stage the data before it is eventually transferred to storage, much similar to a cache. This technique, commonly known now as data staging, relies on allocating additional compute nodes to serve as a transient staging point for output data [36–39].

By relying on system support for remote memory reads, it is possible to schedule data movement from the staging area based on available network bandwidth and staging area memory (Figure 8). This relieves the compute process of all responsibilities beyond the initial creation of the buffer. This

DOI: 10.1002/cpe

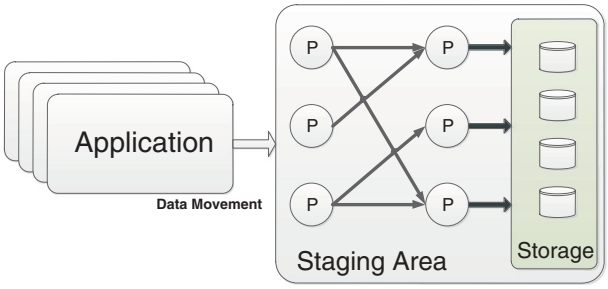


Figure 8. The flow of data in staging.

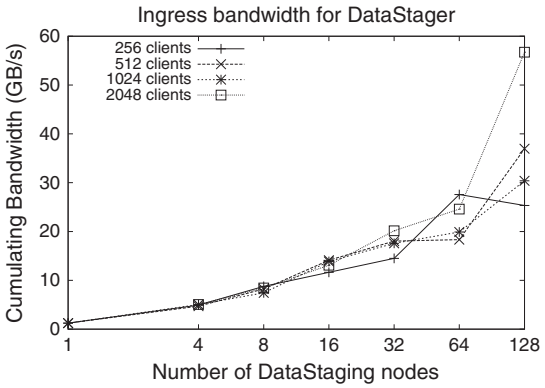


Figure 9. Data movement throughput when using multiple staging nodes. For maximum throughput, we need sufficient number of clients.

pull-based model helps manage the limited memory in the staging area and draws on the model proposed by disk-directed I/O [40] and Panda [41]. The application only sends a small 'data available' request to the staging process. These requests are queued and serviced by a remote direct memory access (RDMA) read as memory becomes available. This server-directed data transfer allows the data to be pulled in over a longer period of time and requires very little participation from the application. Moreover, this approach requires no cross communication between the application processes and requires no synchronization within the application.

In terms of raw throughput, this approach matches and can even exceed the performance for bursty data, the maximum achievable bandwidth of the available system. Consider the throughput shown in Figure 9. Simply by increasing the number of staging servers, the throughput increases as long as the clients themselves are not saturated. Moreover, the variability insulation provided by the buffering in the staging area isolates the application from transient performance failures in the storage back-end.

While asynchronous I/O can insulate the application from transient performance problems, experiments have shown that application runtime would increase significantly when using asynchronous data movement. The effect was magnified as the experiment scaled the application and the output data size. Through extensive experimentation and application analysis, it was discovered that the background data movement was interfering with intra-application communication. In particular, the impact of server side remote reads from application memory would have a substantial negative impact on the performance of collectives.

To address this slowdown, we tested a series of scheduling algorithms that attempted to avoid interference with application communication. The observation that there are regular communication and computation phases in most scientific applications afforded ADIOS' introduction of a dynamic

DOI: 10.1002/cpe

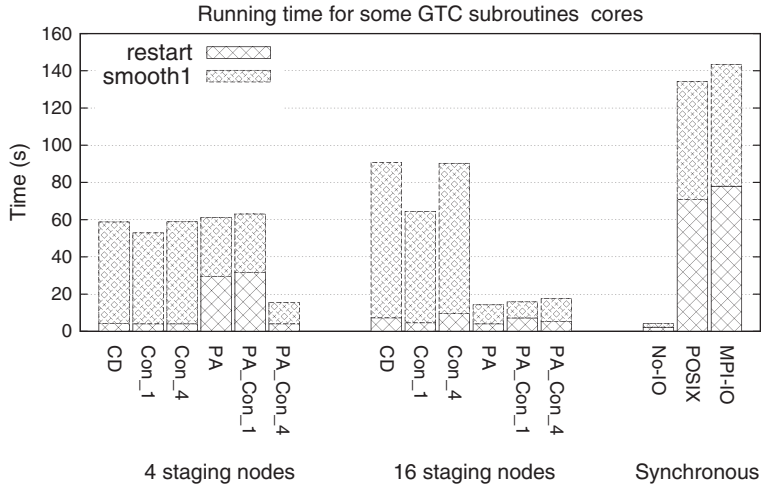


Figure 10. An example of how different scheduling techniques result in vastly different execution times for functions in Gyrokinetic Toroidal Code (GTC).

iterative timeline for application communication. Using this timeline, the server is able to issue data movement requests only for periods where the application was unlikely to be in a communication intensive phase.

Different scheduling schemes to reduce interference are tested using the GTC leadership application on the Oak Ridge Leadership Computing Facility Jaguar system (Figure 10). This showed that due to the complexity of predicting application behavior, a single scheduling method was insufficient for solving the interference problem. Instead a combination of the perturbation avoidance scheduler in combination with a rate limiting scheduler provided the least amount of overhead.

The experiment yielded another insight. Even traditional two phase MPI-IO and standard POSIX FORTRAN writes on Lustre resulted in interference with collective communication. The write operation completed when the local system had buffered the data. While the control returned to the application, Lustre continued to drain the buffer in the background. Thus, the only real method to measuring impact of I/O is to compare performing I/O with performing no I/O at all and taking the difference.

Staging in ADIOS has had a significant impact within the high performance I/O community. There have been new efforts to address the idea of decoupling application performance from storage performance from different view points. Glean [42] and Nessie [18] have sought to address the challenges in scalable data staging on leadership class systems within middleware libraries such as netCDF. Ouyang *et al.* [43] have used SSDs to demonstrate the benefits of data staging for checkpoint performance. Kannan *et al.* [44] demonstrate similar gains for data analytics with non-volatile random-access memory (NVRAM). Similarly, a simplification of the staging concept, burst buffers, have gained traction in attempting to bridge the gaps between high performance simulations and the storage back-end [45]. Utilizing a multi-tier hierarchy of storage that includes SSDs, burst buffers provide a staging area for bursty output from simulations. The data is absorbed into the burst buffer during the write phase, and is streamed out to back-end storage in between writes.

4.6. Application complexity

A major challenge for performing I/O research is accurately representing the I/O behavior of applications. One approach is to instrument a representative application with timing code to measure I/O performance. This technique assures that the performance measurements are relevant because they are acquired directly from the application. However, this approach presents a great deal of unnecessary complexity to an I/O developer because building and running the application typically requires detailed knowledge that is unrelated to the task of I/O performance measurement. Gaining

DOI: 10.1002/cpe

the expertise required to build and execute a scientific application for I/O research is time consuming and poses a major distraction to the process of I/O research.

It is common to measure bulk I/O rates using a tool such as IOR [46] or the NAS parallel benchmark [47, 48]. This makes the testing process less cumbersome, because it is not necessary to deal with the complexities of an application. Unfortunately, the results obtained from this type of benchmark reflect the raw capabilities of the system and may have very little to do with the actual performance of the applications that run on the system.

For these reasons, it is much more common to use *I/O kernels*, such as the FLASH-IO benchmark routine [49] and the S3D I/O kernel [12], to represent applications. I/O Kernels are codes that include the I/O routines from a target application but typically have computation and communication operations removed. This affords quickly testing application I/O behavior without the learning curve and required dependencies of the full application. The use of I/O kernels still presents several problems, including the following: (i) They are seldom kept up-to-date with changes to the target application; (ii) they often retain the same cumbersome build mechanism of the target application; (iii) different I/O kernels do not measure performance the same way; and (iv) an I/O kernel is not necessarily available for every application of interest.

*Input/output skeletal application*s extended the notion of I/O kernels by offering a different paradigm. This is a code that, such as an I/O kernel, includes the same set of I/O operations used by an application while omitting computation and communication. In contrast to I/O kernels, I/O skeletal applications are *automatically generated* from the information included in a high-level I/O descriptor. Skeletal applications share the advantages of I/O kernels while directly addressing the four problems of I/O kernels identified earlier as follows: (i) Because skeletal applications are automatically generated, they are trivial to reconstruct after a change to the application; (ii) all skeletal applications share the same relatively simple build mechanism; (iii) all skeletal applications share the same flexible measurement techniques, making it easier to perform apples to apples comparisons; and (iv) given its I/O descriptor, it is a simple process to generate an I/O skeletal application for any application.

To implement the I/O skeletal application approach, the *Skel* [50] tools support the creation and execution of I/O skeletal applications. Skel consists of a set of python modules, each of which performs a specific task. The *skel-xml* module produces the ADIOS-config file to be used by the skeletal application. The *skel-parameters* module creates a default parameter file based on the configuration file, which is then customized by the user. Modules *ADIOS-config* and *skel-config* interpret the input ADIOS-config file and parameter file, respectively. Based solely on data from those two input files, the *skel-source* module generates the source code, in either C or FORTRAN, that comprises the skeletal application. The *skel-makefile* module generates a makefile for compiling and deploying the skeletal application based on a platform specific template. Finally, the *skel-submit* module generates a submission script, also based on a template, suitable for executing the skeletal application on the target platform.

To enhance the measurements offered by Skel, ADIOS has been extended with an optional mechanism for providing low-level timing information and is collected on each application process. The ADIOS I/O methods of interest have been customized with timing instructions using the timing mechanism. The details of the low level timing operations vary according to the I/O method in use and have been designed to support arbitrary measurements that may be needed by new I/O methods added in the future.

This combination of easy-to-use, automatically generated I/O benchmarks along with finer-grained timing of I/O methods has proven to be quite powerful. By allowing application I/O patterns to be tested quickly without the need to build and run the full application a preliminary view of the potential improvements to an application's performance can be determined ahead of time showing whether additional optimization efforts are warranted. In many cases, the I/O performance of skeletal applications has closely matched application I/O performance. It is still useful to validate the performance measured by the skeletal applications by comparing some measurements with performance of the original application. In one case, a descrepancy when comparing the performance of a FORTRAN application with that of a C skeletal application surfaced, despite both codes using the same underlying I/O library. The problem was eliminated by ensuring that the skeletal applications

DOI: 10.1002/cpe

are generated using the same language as the target application. In another instance, a large discrepancy between application and skeletal was due to the measurement performed in the application encompassed additional computation in addition to the I/O calls of interest. Finally, variations in measurement technique are often significant, including when barrier operations are used, and whether a measurement is taken from a single process or reflects a reduction operation across all processes.

We currently bundle several ADIOS-config files with skel, but are planning to extend this by making a standard set of full skel benchmarks available on our website, along with performance results from a variety of machines.

4.7. Interleaving data compression and parallel I/O

A promising solution to the I/O bottleneck is to perform data reduction. Unfortunately, state-of-the-art I/O middleware solutions currently do not have native support for write compression in a parallel context, due to the complexity of handling the resultant varying sizes of the compressed data buffers that requires synchronization between all nodes performing shared-file I/O. Furthermore, data reduction that sacrifices simulation fidelity, especially at checkpoints, rules out lossy compression as a generally viable data reduction method. And yet, typical lossless compression techniques are ineffective on hard-to-compress floating-point data generally produced by such simulations.

Arguably, interleaving lossless compression and parallel I/O is a 'secret sauce' for addressing the I/O bottleneck [51]. Using dynamic, fast identification of highly compressible bytes to process by a lossless compression method, such as ISOBAR [52], while asynchronously writing the remaining, uncompressible bytes to storage with ADIOS can effectively hide the cost of compression and I/O synchronization behind this transfer, thus rendering parallel write compression viable.

Such an interleaving method naturally fits into data staging architectures, where various data transformations can occur while data is 'in transit' or *in situ*, from compute nodes to disk. Traditionally, staging has been used to compute statistical analyses or perform indexing operations [38, 53]. With interleaved compression and I/O, however, this functionality can be augmented by performing compression as a *in-situ* transfer and storage optimization, as well.

This system exhibits both read and write performance gains proportional to the degree of data reduction, which ranges as high as 46% on scientific datasets, in addition to reducing the total amount of data being stored and accessed. Without interleaving, by merely compressing the data and then writing it to storage, only marginal gains, as high as 6%, could be attained. Because ISOBAR applies a preconditioner to identify those byte columns that are 'compressible', it avoids wasting CPU cycles trying to compress incompressible bytes in the data. ISOBAR groups the compressible parts together as small less than 3 MB chunks and performs compression only on those parts. By operating on a lower memory footprint and in an embarrassingly parallel manner, this technique results in higher energy-efficiency while simultaneously offering high throughput, reduced data movement, and data reduction that collectively translate to a $1.8 - 5.5\times$ reduction in energy consumption.

4.8. In transit indexing

As data sizes grow, the naive approach of answering user questions by reading and examining each data record can be prohibitively expensive. An effective way to directly access 'interesting' records is by using an index for the data [54]. In existing data analysis environments, such as the widely used database management systems, the user data is loaded and written to disk first before the indexes are created. In this case, the first step of creating an index is to read user data into memory from disk. Because the read operation is often the most time consuming part of the index creation process, this reading cost can be avoided by creating indices, whereas the data is being written to disk, that is, in transit indexing [55]. This approach has the added advantage that the indices will be available as soon as the data is available, which can further promote the use of indices for data analyses.

Designing, implementing, and performance measurements for indices have yielded a number of valuable lessons. The top three lessons are as follows:

DOI: 10.1002/cpe

Avoiding synchronization. A careful analysis showed that synchronization in the writing step of index construction increases with the number of processors used. This growth leads the total index construction time to increase more as a larger number of processors are used. Redesigning the index storage system to avoid this synchronization is needed in the future.

Index organization. Store the indices with the user data in the same file. This is a convenient choice, but it does have performance implications. For example, it makes the synchronization more necessary. One way to avoid this synchronization would be bring indexing operations inside the ADIOS system itself.

Choosing a moderate number of processors. Using more processors is useful for reducing the time needed for building indexes and answering queries. However, it is not necessary to choose the largest number of processors possible. A modest number of processors, say 32, typically gives good performance.

4.9. Provenance and access pattern mining

Adaptable IO System provenance information, a collection of metadata related with user file access activities (open, read, write, etc) through ADIOS, is getting more attention. Obtaining detailed data access information (e.g., accessing variables inside a file or dimensions of variables) is difficult through a native file system because all of the scientific data file formats (i.e., HDF, NetCDF, and ADIOS BP) that are frequently used have their own data formats and (hierarchical) structures that are independent from the native file system. ADIOS offers an ideal platform on which to base the collection of standard provenance data. Once collected, provenance information can be used in mining and discovering access patterns or hidden knowledge that can be used for various purposes. For example, we are currently developing access pattern mining techniques that can be used to build an efficient data pre-fetcher to predict a user's data requirements and load data before the request in order to reduce data access latency.

5. FURTHER CHALLENGES

The ADIOS framework successfully solves some of the current challenges facing leadership applications today. The next generation of Exascale architectures and applications will provide a new host of challenges. In order to provide a graduated research and development cycle as well as to aid the reader's understanding of the next generation, the major research goals below are partitioned into two time frames. The first Exascale systems are projected to be deployed around 2018 at leadership computing facilities at Oak Ridge National Laboratory and Argonne National Laboratory. This is the time frame for the first set of challenges addressing the needs of leadership capability applications running on millions of cores and producing petabytes of data per hour. The next major shift will come in about 10 years when Exascale computing will become more commoditized. The needs will shift from capability computing to mixture of capability and capacity computing.

For the initial time scale of 5–7 years, the following are some of the significant goals for ADIOS.

1. *Data integrity.* Integrity is projected to become a significant concern for scientific data as platforms and architectures transition toward Exascale. Even today, leadership applications can produce hundreds of terabytes of data per day and are projected to increase by three orders of magnitude in the next decade [2]. Although some data integrity issues can be managed through a filesystem layer through checksum and block hashing such as is performed by the Z file system (ZFS) [56], integrity without higher level application knowledge will be both expensive and incomplete. As the number of components involved in computations increases, the probability of error at each of the various data generation points also increases. Relying only on the file system to provide integrity checks does not address data corruption as data moves from the processor cache to memory to disk. Instead, the higher level mechanism that

DOI: 10.1002/cpe

ADIOS provides today can be extended to include, alongside data definitions and descriptions, information about expected ranges of variables. Using the I/O pipeline concept described earlier, ADIOS can provide higher level integrity protection using semantic knowledge. This can also be used in combination with file system level integrity checks for additional protection.

2. *Fault detection and recovery.* Even more than the issue of data integrity, Exascale will require a high level of resilience to faults both in the hardware and in the software. The increased architectural complexity of the many-core approach coupled with the estimated million-way concurrency will create a significant risk of both transient failures and permanent faults in the hardware. For an I/O framework to provide application support for Exascale scientific applications, it must provide both application-level and system-level hooks for fault detection and recovery. Due to the wide range of faults, the use of application-level knowledge about data will be a critical mechanism for providing fault resilience. ADIOS is in a unique position to provide fault detection and recovery for Exascale applications due to the separation of data description from data layout. Thus, ADIOS can easily incorporate new methods for extending resilience without requiring substantial refactoring of application code.
3. *Automated performance tuning.* As demonstrated by the utility of skel described in Section 4.6, providing utilities that afford easier performance measurement without dealing with the complexities of applications will lead to a big improvement in productivity. As Exascale platforms are introduced, this capability will need to extend to automated performance tuning in addition to performance capture and analysis. The experience with skel must be combined with contemporary research in profiling for I/O systems as well as auto-tuning frameworks to achieve this goal. Combining this capability with the collection of power metrics can help optimize data movement from applications to reduce energy use while improving time to completion.
4. *Managed data consistency.* The last, and perhaps the biggest research goal for ADIOS in the Exascale age, will be to introduce and create flexible data consistency semantics. The challenge is not just technical. There is strong inertia among application scientists and system administrators toward requiring strong consistency semantics. However, as new Exascale hardware scales toward million-way concurrency, maintaining a tightly consistent view of data between even a fraction of the available cores will be nearly impossible. Instead of relying on unviable strong consistency semantics provided by the file system, ADIOS must attempt to offer a more continuous consistency model allowing the developers to choose to sacrifice performance for tighter consistency or vice versa. Similar to the flexibility offered on a continuity scale, ADIOS must be able to provide a broad range of consistency for a variety of data sets. Moreover, global consistency will often be unnecessary for large scale applications. Instead, regional consistency between neighbors will need to be provided to allow acceptable performance.

As Exascale becomes more commoditized, the next set of challenges will require techniques for wide-scale data availability by combining data centers with cloud storage and even mobile access to data. New methods for data reduction and flexible compute placement will be needed to deal with this challenge. Research in this time frame will be to investigate methods for heterogenous I/O platforms where the cost, power usage, and performance characteristics vary across a wide set of metrics. The complexity of such a system will place significant pressure on higher level I/O frameworks to improve both usability and maintainability. Providing feedback to the user in a manner that does not overwhelm will be a crucial goal for I/O research in this time period.

The ADIOS team is currently exploring many techniques for addressing these goals and strives to provide leadership in this area over the next decade while also providing developers and users with an easy-to-use, high performance infrastructure for data management.

ACKNOWLEDGEMENTS

The authors would like to thank Garth Gibson and Rob Ross of Carnegie Mellon University and Argonne National Laboratory for their extensive early input. Their shared experiences with parallel I/O helped

DOI: 10.1002/cpe

eliminate many dead ends in the early ADIOS system. We would also like to thank Sean Ahern, Ilkay Altintas, Micah Beck, John Bent, Luis Chacon, C.S. Chang, Jackie Chen, Hank Childs, Julian Cummings, Divya Dinkar, Ciprian Docan, Greg Eisenhauer, Stepane Ethier, Ray Grout, Steve Hodson, Chen Jin, Ricky Kendall, Todd Kordenbrock, Seong-Hoe Ku, Tahsin Kurc, Sriram Lakshminarasimhan, Wei-keng Liao, Zhihong Lin, Xiaosong Ma, Ken Moreland, D.K. Panda, Valerio Pascucci, Milo Polte, Dave Pugmire, Joel Saltz, Ramanan Sankaran, Mladen Vouk, Fang Zheng, and Fan Zhang who worked with our team to contribute many useful suggestions.

REFERENCES

1. Lofstead J, Klasky S, Schwan K, Podhorszki N, Jin C. Flexible IO and integration for scientific codes through the Adaptable IO System (ADIOS). *Clade 2008 at HPDC*, Boston, Massachusetts: ACM, June 2008. [Online]. (Available from: http://www.adiosapi.org/uploads/clade110-lofstead.pdf) [Accessed on August 14, 2013].
2. Scientific discovery at the exascale: report from the DOE ASCR 2011 workshop on exascale data management, analysis and visualization. [Online]. (Available from: http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Exascale-ASCR-Analysis.pdf) [Accessed on August 14, 2013].
3. Thakur R, Gropp W, Lusk E. On implementing MPI-IO portably and with high performance. *Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems*, Atlanta, Georgia, USA, ACM, 1999; 23–32.
4. Yu W, Vetter J, Oral H. Performance characterization and optimization of parallel I/O on the cray XT, *IPDPS*, pp.1–11, April 2008.
5. The HDF Group. Hierarchical data format version 5, 2000–2010. (Available from: http://www.hdfgroup.org/HDF5) [Accessed on August 14, 2013].
6. Li J, Liao W-K, Choudhary A, Ross R, Thakur R, Gropp W, Latham R, Siegel A, Gallagher B, Zingale M. Parallel netCDF: a high-performance scientific I/O interface. *Proc. SC03*, Phoenix, Arizona, USA, ACM, 2003. (Available from: http://www.mcs.anl.gov/parallel-netcdf) [Accessed on August 14, 2013].
7. NetCDF. (Available from: http://www.unidata.ucar.edu/software/netcdf/) [Accessed on August 14, 2013].
8. Unidata. (Available from: http://www.hdfgroup.org/projects/netcdf-4/) [Accessed on August 14, 2013].
9. Bent J, Gibson G, Grider G, McClelland B, Nowoczynski P, Nunez J, Polte M, Wingate M. Plfs: a checkpoint filesystem for parallel applications. *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09. New York, NY, USA: ACM, 2009; 21:1–21:12. [Online]. (Available from: http://doi.acm.org/10.1145/1654059.1654081) [Accessed on August 14, 2013].
10. Mehta K, Bent J, Torres A, Grider G, Gabriel E. A plugin for HDF5 using PLFS for improved I/O performance and semantic analysis. *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion*, Salt Lake City, Utah, USA, IEEE, 2012; 746–752.
11. Klasky S, Ethier S, Lin Z, Martins K, McCune D, Samtaney R. Grid-based parallel data streaming implemented for the gyrokinetic toroidal code. *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, Phoenix, Arizona, USA, 2003.
12. Chen J, Choudhary A, De Supinski B, DeVries M, Hawkes E, Klasky S, Liao W, Ma K, Mellor-Crummey J, Podhorszki N, Sankaran R, Shende S, Yoo CS. Terascale direct numerical simulations of turbulent combustion using S3D. *Computational Science & Discovery* 2009; **2**:015001.
13. Oldfield RA, Maccabe AB, Arunagiri S, Kordenbrock T, Riesen R, Ward L, Widener P. Lightweight I/O for scientific applications. *Proceedings of the IEEE International Conference on Cluster Computing*, Barcelona, Spain, September 2006. [Online]. (Available from: http://doi.ieeecomputersociety.org/10.1109/CLUSTR.2006.311853) [Accessed on August 14, 2013].
14. Tchoua R, Abbasi H, Klasky S, Liu Q, Podhorszki N, Pugmire D, Tian Y, Wolf M. Collaborative monitoring and visualization of HPC data. *2012 International Symposium on Collaborative Technologies and Systems (CTS)*, Denver, Colorado, USA, May 2012; 397–403.
15. Lofstead J, Polte M, Gibson G, Klasky S, Schwan K, Oldfield R, Wolf M, Liu Q. Six degrees of scientific data: reading patterns for extreme scale science IO. *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11. New York, NY, USA: ACM, 2011; 49–60. [Online]. (Available from: http://doi.acm.org/10.1145/1996130.1996139).
16. Chen JH, Choudhary A, De Supinski B, DeVries M, Hawkes E, Klasky S, Liao W, Ma K, Mellor-Crummey J, Podhorszki N, Sankaran R, Shende S, Yoo sCS. Terascale direct numerical simulations of turbulent combustion using S3D. *Computational Science & Discovery* 2009; **2**(1):015001 (31pp). [Online]. (Available from: http://stacks.iop.org/1749-4699/2/015001).
17. Yu W, Vetter J. ParColl: partitioned collective I/O on the cray XT. *International Conference on Parallel Processing* 2008; **0**:562–569.
18. Lofstead J, Oldfiend R, Kordenbrock T, Reiss C. Extending scalability of collective I/O through Nessie and staging. *The Petascale Data Storage Workshop at Supercomputing*, Seattle, WA, November 2011; 7–12.
19. Alverson R, Roweth D, Kaplan L. The Gemini system interconnect. *Proceedings of the 18th Annual Symposium on High Performance Interconnects (HOTI)*, Mountain View, CA: IEEE Computer Society Press, August 2010; 83–87.
20. Lofstead J, Zheng F, Klasky S, Schwan K. Adaptable, metadata rich IO methods for portable high performance IO. *2009. IPDPS 2009. IEEE International Symposium on Parallel Distributed Processing*, May 2009; 1 –10.

DOI: 10.1002/cpe

21. Wang WX, Lin Z, Tang WM, Lee W, Ethier S, Lewandowski JLV, Rewoldt G, Hahm TS, Manickam J. Gyro-kinetic simulation of global turbulent transport properties in Tokamak experiments. *Physics of Plasmas* 2006; **13**(9):092505. [Online]. (Available from: http://link.aip.org/link/?PHP/13/092505/1) [Accessed on August 14, 2013].
22. Chang CS, Klasky S, Cummings J, Samtaney R, Shoshani A, Sugiyama L, Keyes D, Ku S, Park G, Parker S, Podhorszki N, Strauss H, Abbasi H, Adams M, Barreto R, Bateman G, Bennett K, Chen Y, Azevedo ED, Docan C, Ethier S, Feibush E, Greengard L, Hahm T, Hinton F, Jin C, Khan A, Kritz A, Krsti P, Lao T, Lee W, Lin Z, Lofstead J, Mouallem P, Nagappan M, Pankin A, Parashar M, Pindzola M, Reinhold C, Schultz D, Schwan K, Silver D, Sim A, Stotler D, Vouk M, Wolf M, Weitzner H, Worley P, Xiao Y, Yoon E, Zorin D. Toward a first-principles integrated simulation of Tokamak edge plasmas - art. no. 012042. *Scidac 2008: Scientific Discovery through Advanced Computing* 2008; **125**:12042–12042.
23. Cui Y, Olsen KB, Jordan TH, Lee K, Zhou J, Small P, Roten D, Ely G, Panda DK, Chourasia A, Levesque J, Day SM, Maechling P. Scalable earthquake simulation on petascale supercomputers. *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10. Washington, DC, USA: IEEE Computer Society, 2010; 1–20. [Online]. (Available from: http://dx.doi.org/10.1109/SC.2010.45).
24. Chacón L. A non-staggered, conservative, $\nabla \dot{s} \boldsymbol{B} \rightarrow = \mathbf{0}$, finite-volume scheme for 3D implicit extended magnetohydrodynamics in curvilinear geometries. *Computer Physics Communications* 2004; **163**:143–171.
25. Pretorius F. Evolution of binary black hole spacetimes. *Physical Review Letters* 2005; **95**(121101):1–4.
26. Messer OEB, Bruenn SW, Blondin JM, Hix WR, Mezzacappa A, Dirk CJ. Petascale supernova simulation with CHIMERA. *Journal of Physics Conference Series* 2007; **78**(1):1–5.
27. Grout RW, Gruber A, Yoo C, Chen J. Direct numerical simulation of flame stabilization downstream of a transverse fuel jet in cross-flow. *Proceedings of the Combustion Institute* 2011; **33**(1):1629–1637.
28. Childs H. Architectural challenges and solutions for petascale postprocessing. *Journal of Physics* 2007; **78**(012012).
29. Ding CHQ, He Y. Data organization and I/O in a parallel ocean circulation model. *Proc. SC99*, Portland, Oregon, USA, Society Press, 1999.
30. SDSC Gordon: Data-Intensive Supercomputing. (Available from: http://gordon.sdsc.edu) [Accessed on August 14, 2013].
31. Sarawagi S, Stonebraker M. Efficient organization of large multidimensional arrays. *Proc. 10th Int. Conf. on Data Eng.*, Houston, TX, 1994; 328–336.
32. Sagan H, Holbrook J. *Space Filling Curves*. Springer-Verlag: New York, NY, USA, 1994.
33. Hilbert D. Ueber die stetige abbildung einer line auf ein flächenstück. *Mathematische Annalen* 1891; **38**:459–460.
34. Moon B, Jagadish H, Faloutsos C, Saltz J. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering* 2001; **13**(1):124–141.
35. Borrill J, Oliker L, Shalf J, Shan H. Investigation of leading HPC I/O performance using a scientific-application derived benchmark. *Proceedings of the 2007 Conference on Supercomputing, SC07*, Reno, Nevada, USA, 2007.
36. Docan C, Parashar M, Klasky S. Dataspaces: an interaction and coordination framework for coupled simulation workflows. *IEEE International Symposium on High Performance Distributed Computing*, 2010; 25–36.
37. Docan C, Parashar M, Klasky S. Enabling high speed asynchronous data extraction and transfer using dart. *Concurrency and Computation: Practice and Experience* 2010; **22**(9):1181–1204.
38. Abbasi H, Wolf M, Eisenhauer G, Klasky S, Schwan K, Zheng F. DataStager: scalable data staging services for petascale applications. *Proceedings of the 18th International Symposium on High Performance Distributed Computing*, Garching, Germany, HPDC '09. ACM, 2009; 39–48. [Online]. (Available from: http://doi.acm.org/10.1145/1551609.1551618) [Accessed on August 14, 2013].
39. Abbasi H, Lofstead JF, Zheng F, Schwan K, Wolf M, Klasky S. Extending I/O through high performance data services. *CLUSTER*, New Orleans, Louisiana, USA, IEEE, 2009; 1–10.
40. Kotz D. Disk-directed I/O for MIMD multiprocessors. *ACM Transactions on Computer Systems* 1997; **15**(1):41–74. [Online]. (Available from: http://www.cs.dartmouth.edu/~dfk/papers/kotz:jdiskdir.ps.gz).
41. Seamons KE, Winslett M. Physical schemas for large multidimensional arrays in scientific computing applications. *Proceedings of the 7th International Working Conference on Scientific and statistical Database Management*, Charlottesville, Virginia, USA, September 1994; 218–227. [Online]. (Available from: http://bunny.cs.uiuc.edu/CADR/pubs/ssdbm.ps) [Accessed on August 14, 2013].
42. Vishwanath V, Hereld M, Papka ME. Toward simulation-time data analysis and I/O acceleration on leadership-class systems. *2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, Davis, California, USA, IEEE, 2011; 9–14.
43. Ouyang X, Marcarelli S, Panda DK. Enhancing checkpoint performance with staging io and SSD. *2010 International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, Lake Tahoe, Nevada, USA, IEEE, 2010; 13–20.
44. Kannan S, Gavrilovska A, Schwan K, Milojicic D, Talwar V. Using active NVRAM for I/O staging. *Proceedings of the 2nd International Workshop on Petascale Data Analytics: Challenges and Opportunities*, Seattle, Washington, USA, ACM, 2011; 15–22.
45. Liu N, Cope J, Carns P, Carothers C, Ross R, Grider G, Crume A, Maltzahn C. On the role of burst buffers in leadership-class storage systems. *2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, Pacific Grove, California, USA, IEEE, 2012; 1–11.
46. IOR HPC benchmark. (Available from: http://sourceforge.net/projects/ior-sio/) [Accessed on August 14, 2013].

DOI: 10.1002/cpe

47. Bailey D, Harris T, Saphir W, Van Der Wijngaart R, Woo A, Yarrow M. The NAS parallel benchmarks 2.0. *Technical Report NAS-95-020*, NASA Ames Research Center, 1995.
48. Bailey DH, Barszcz E, Barton JT, Browning DS, Carter RL, Dagum L, Fatoohi RA, Frederickson PO, Lasinski TA, Schreiber RS, Simon HD, Venkatakrishnan V, Weeratunga SK. The NAS parallel benchmarks summary and preliminary results. *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing, 1991. Supercomputing'91*, Albuquerque, New Mexico, USA, IEEE, 1991; 158–165.
49. FLASH I/O benchmark routine – parallel HDF5. (Available from: http://www.ucolick.org/~zingale/flash_benchmark_io/) [Accessed on August 14, 2013].
50. Logan J, Klasky S, Lofstead J, Abbasi H, Ethier S, Grout R, Ku S-H, Liu Q, Ma X, Parashar M, Podhorszki N, Schwan K, Wolf M. Skel: generative software for producing skeletal I/O applications. *The Proceedings of D^3science*, Stockholm, Sweden, 2011.
51. Schendel E, Pendse S, Jenkins J, Boyuka II D, Gong Z, Lakshminarasimhan S, Liu Q, Kolla H, Chen J, Klasky S, Ross R, Samatova N. ISOBAR hybrid compression-I/O interleaving for large-scale parallel I/O optimization. *Proceedings of the 21st International Symposium on High Performance Distributed Computing*, HPDC '12. New York, NY, USA: ACM, 2012; 61–72.
52. Schendel ER, Jin Y, Shah N, Chen J, Chang CS, Ku S-H, Ethier S, Klasky S, Latham R, Ross R, Samatova NF. ISOBAR preconditioner for effective and high-throughput lossless data compression. *IEEE 28th International Conference on Data Engineering (ICDE)*, Washington, DC, USA, 2012; 138–149.
53. Abbasi H, Eisenhauer G, Wolf M, Schwan K, Klasky S. Just in time: adding value to the IO pipelines of high performance applications with JITStaging, Proceedings of the 20th International Symposium on High Performance Distributed Computing, San Jose, California, USA, HPDC '11. ACM, 2011; 27–36. [Online]. (Available from: http://doi.acm.org/10.1145/1996130.1996137) [Accessed on August 14, 2013].
54. Wu K, Ahern S, Bethel EW, Chen J, Childs H, Cormier-Michel E, Geddes C, Gu J, Hagen H, Hamann B, Koegler W, Lauret J, Meredith J, Messmer P, Otoo E, Perevoztchikov V, Poskanzer A, Prabhat OR, Shoshani A, Sim A, Stockinger K, Weber G, Zhang W-M. FastBit: interactively searching massive data. *SciDAC*, San Diego, California, USA, 2009.
55. Kim J, Abbasi H, Chacón L, Docan C, Klasky S, Liu Q, Podhorszki N, Shoshani A, Wu K. Parallel in situ indexing for data-intensive computing. *LDAV*, Davis, California, USA, IEEE, 2011; 65–72.
56. Zhang Y, Rajimwale A, Arpaci-Dusseau AC, Arpaci-Dusseau RH. End-to-end data integrity for file systems: a ZFS case study. *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST'10. Berkeley, CA, USA: USENIX Association, 2010; 3–3. [Online]. (Available from: http://dl.acm.org/citation.cfm?id=1855511.1855514) [Accessed on August 14, 2013].

DOI: 10.1002/cpe