In situ data processing for extreme-scale computing

Scott Klasky[1], Hasan Abbasi[2], Jeremy Logan[1], Manish Parashar[6], Karsten Schwan[4], Arie Shoshani[3], Matthew Wolf[4], Sean Ahern[1], Ilkay Altintas[9], Wes Bethel[3], Luis Chacon[1], CS Chang[10], Jackie Chen[5], Hank Childs[3], Julian Cummings[13], Ciprian Docan[6], Greg Eisenhauer[4], Stephane Ethier[10], Ray Grout[7], Sriram Lakshminarasimhan, Zhihong Lin[10], Qing Liu[2], Xiaosong Ma[15], Kenneth Moreland[5], Valerio Pascucci[12], Norbert Podhorszki[1], Nagiza Samatova[15], Will Schroeder[8], Roselyne Tchoua[1], Yuan Tian[14], Raju Vatsavai[1], John Wu[3], Weikuan Yu[14], Fang Zheng[4]

[1]ORNL, [2] U.T. Knoxville, [3]LBNL, [4]Georgia Tech, [5]Sandia Labs, [6] Rutgers, [7]NREL, [8]Kitware, [9]UCSD, [10]PPPL, [11]UC Irvine, [12]U. Utah, [13] Caltech, [14]Auburn University, [15]NCSU

Abstract

The U.S. Department of Energy established leadership-computing facilities in 2004 to provide scientists with capability computing for high-profile science. Since inception, the system capacity has grown from 14 TF to 1.8 PF, an increase of more than a factor of 100, and it will increase by another factor of 100 in the next five years. This growth, along with computing policies that enable users to run at scale for long periods, has allowed scientists to write unprecedented amounts of data to the file system. At the same time, the effective speed of the I/O system (time to write full system memory to the file system) has decreased, going from 350 seconds on ASCI Purple (49 TB at 140 GB/s) to 1500 seconds on Jaguar (300 TB at 200 GB/s). Because future systems will further intensify this imbalance, we need to extend the definition of I/O to include that of I/O pipelines, which blend together self-describing file formats, staging methods with visualization and analysis "plugins," and new techniques to multiplex outputs using a novel service-oriented architecture approach—all in an easy-to-use I/O componentization framework that can add resiliency to large-scale calculations. Our approach to this has been via the community-created ADIOS system, which is driven by many of the leading-edge application teams and the top I/O researchers.

1 Introduction

Over the past decade the capability of leadership-computing systems has increased substantially, with the Top500 machines seeing a many-fold increase from a few teraflops to more than a petaflop for the current generation of high-performance computing (HPC) systems. This increase in compute power has not been mirrored in I/O performance, which instead has suffered by comparison and even decreased in absolute terms. This imbalance, coupled with the increase in data volumes produced by applications, has resulted in a new bottleneck in the performance of these systems.

Conventional methods for addressing this bottleneck, such as increasing I/O backend capability by adding more disks with higher speeds, are unlikely to keep up with the performance issues due to the costs associated with storage. Because I/O performance has little effect on procurement for next-generation machines, the performance of the I/O system is unlikely to match the growth of compute capability in the future. The problem is further exacerbated by the inefficiency of I/O performance; most applications are unable to achieve a significant fraction of the peak performance of the storage system [1]. This is due mostly to the complexity of traditional I/O methods, where the developer must make a heroic effort to optimize the application I/O. This limit on usability directly impacts the possible performance of the application.

Extended memory hierarchies will once again further complicate the I/O performance for many applications. The conventional wisdom will be to simply write data to the nonvolatile

memory, thereby creating a buffer for the application from the actual file system. The problems that arise are numerous if one simply does the "easy" thing, which is to write the data and forget about it. For one thing, the data must eventually end up on the file system, and this data movement will be shared with other data movement on the HPC, including other applications writing and reading from the file system. Another problem is that overall application performance can suffer if the data movement is not scheduled properly with the applications. Applications will need to learn different "tricks" to attain acceptable levels of I/O performance. Additionally, applications will become increasingly reliant on in situ visualization techniques in order to reduce the penalties associated with storage and retrieval of data.

In this paper we introduce a conceptual shift in addressing I/O performance bottlenecks by addressing the needs of extreme-scale computing. To achieve appropriate performance, we propose a multipoint approach to handling some of the biggest concerns and demonstrate a framework that has been developed with the following design guidelines:

1. **Service-oriented architecture (SOA).** The overarching framework should be easy for new programmers to write individual "plugins" that can be compiled and either run independently, by reading data from the file system, or work in situ, where users can introduce their services at run time. In this way, developers have the flexibility to independently develop their executable, such as a visualization program. Furthermore, by creating a separate executable, applications are not littered with extra lines of code for extending the functionality of the code through these services. Services should be discoverable to one another and allow for efficient communication (either memory-to-memory or one-sided memory transfers).
2. **In situ processing.** The increasing volume of data generated by the applications has added constraints on how easily and efficiently it can be processed. Requiring that the application move the data to disk before process completion is not a viable mechanism at extreme scale. Instead, in situ or in-place processing has become an important tool for dealing with data. By processing data using domain-specific services or plugins without incurring any additional data movement, we can start to manage the costs of I/O.
3. **Data staging.** Staging is quickly becoming an important foundation of extreme-scale I/O by providing both a staging point for data on its way to storage and the computing resources to initiate in-transit and in situ operations on the data. Data staging complements both the service-oriented architecture for I/O pipelines and in situ processing plugins. The use of staging resources for processing the data in preparation for analysis (as demonstrated in PreData [2]) or for in situ visualization provides a powerful platform for managing I/O at extreme scale.
4. **Data management.** The volume of data projected from exascale applications cannot efficiently be sent to disk without incurring a large performance overhead. Reduction methods such as compression and multiresolution output can decrease this volume to manageable levels without impacting either data viability or application performance. In one such effort, ISABELA [3], which deals with data compression for high-entropy particle data, we envision that data will need to be output and stored through a multiresolution method to expedite the in situ processing of high-resolution scientific data. In addition, data retrieval has become a concern for scientists, especially as the data sets grow to petascale and beyond. Efficient access to data will increasingly require the aid of indices not only to speed up access but also to reduce the volume of the input

data. Techniques such as adding characteristics to the data at output time, using bitmap indexing as in FastBit [4], are required to retrieve the data efficiently for analysis and visualization.

5. **Monitoring.** The above goals are complemented by low-overhead, near-real-time monitoring of the application, allowing users to be aware of the changing landscape of the simulation and make decisions on how the data is handled by the system. We present the eSiMon [5] system for easy monitoring and control of applications.
6. **Usability of optimizations.** In addition to the above, the I/O framework is responsible for providing an easy-to-use interface. The complexity of past interfaces has provided a counterpoint that forces us to rethink how the interface is designed. Instead of tightly integrating the optimizations into the application, we propose the idea of component I/O methods that deal with system-level optimizations in which only the data description and output routines are placed in the application source. This also lets us gain a measure of portability for the application. Moving between different architectures simply requires switching to the appropriate method.
7. **Data formats.** Dealing with extreme-scale data requires a new generation of data formats optimized for massively parallel machines and specifically targeted for scientific data. File formats should be self-describing and provide resiliency and performance at extreme scale. Furthermore, the file format must be amenable to optimizations for both read and write performance. By understanding many of the common reading patterns, we can optimize the file format for the majority of patterns, reducing the typical optimization bias to which some file formats fall prey. In this paper we present the ADIOS-BP file format, which exemplifies these goals.

With the aforementioned concerns in mind, we designed the ADIOS [6] I/O framework not only to address the system I/O issues but also to provide an easy-to-use mechanism that allows scientific developers to work from I/O skeleton applications. Through the use of an optional metadata file, which describes the output data and enables automatic generation of the output routines, the burden on the user is reduced substantially. ADIOS componentizes different methods of I/O, allowing the user to easily select the optimal method. In concert with data staging, this work exemplifies what the next generation of I/O frameworks should look like.

2 Application use cases

Large-scale simulations run on the leadership-computing facilities by application scientists, such as those of the Center for Plasma Edge Simulation (CPES), the Sandia Combustion Research Facility, the Gyrokinetic Simulation of Energetic Particle Turbulence and Transport, and the Pixie3D code, are already generating enormous amounts of data. These simulations are solving complex PDEs to unravel many of the mysteries in science, and they are writing large datasets frequently for postprocessing and for checkpoint-restart capabilities. Typically, these simulations need to write over 100 TB of data per 24 hours of wall-clock time, but often they are limited by severe constraints on the I/O, visualization, and analysis systems. By working directly with the aforementioned application teams and many others, we have learned that as these simulations increase in their spatiotemporal resolution, the scientists will be severely limited in their understanding of the data they produce. This realization forces our community to look for revolutionary changes in the way scientists interact with their data.

Today, data exploration mostly occurs through postprocessing tools, such as VisIt and ParaView, and quite often I/O is the bottleneck. In fact, many visualization scientists have documented that over 50% of the time spent in real-time visualization is spent in I/O, and these numbers are worsening exponentially because of the dramatic changes brought about by employing powerful acceleration technologies such as GPGPUs. Reducing the I/O bottleneck requires the integration of the application and the visualization library, a technique that may not scale and introduces concerns for resiliency.

Application scientists are also constrained with respect to I/O on today's machines by data management and visualization issues, such as the variability of I/O performance, as well as by the lack of optimizations for typical reading patterns used during analysis and visualization. Because ad hoc techniques are often used to determine when to produce output and where to store output data, and because the data volume is overwhelming, researchers never examine much of the data in many spatiotemporal regions. Therefore, new techniques are needed to gain insight into the data and to reduce the total amount of data generated without reducing the quality of the output.

One typical use case comes from PIC codes such as XGC1, GTC, and GTS, which contain hundreds of billions of particles that need to be processed to gain new insights into the data. Scientists may output only a much smaller subset of the data at uniform frequency for post-processing. The problem is that these reductions in both time and space are often determined by user selection, and not by scientific criterion or real-time monitoring of the data. Even so, by writing out 1/100 of the data at 1/10 of the time steps, over 10 TB of data can be generated in one day. After the simulation, the data needs to be archived and analyzed. By using visualization techniques in VisIt and by querying the particles, the scientists can start to determine interesting new physics from their data. However, as we increase the resolution of the data and move to the exascale, the total amount of data quickly becomes unmanageable.

Similarly with Pixie3D [7–9], the scientist sees that the I/O time in the code can become a significant portion of the total time. Previously, Pixie3D would write data, one file per process, and concatenate the data for each time step in each separate file. After the simulation, the Pixplot code would read in these files and write new files in HDF5, which would then be read in by VisIt for analysis and post-processing. Typically, VisIt expressions would allow less data to be written, so that new quantities were generated while they were being read in. The problem with this approach was that the initial files were not self-describing and they took a substantial amount of time to write. The original Pixplot code would later run on either the same number of processors as Pixie3D or in serial to create HDF5 files, which would also take a very long time, although these steps were automated by our team via the Kepler workflow system so that the user could allow the workflow system to manage the data.

In the foreseeable future, our data needs will continue to climb beyond the ability of current tools to cope with them. Simulations from some of the most important science areas in the DOE portfolio are continuing to push the boundaries of the HPC systems, and it has become clear that scientists will need to more carefully monitor and manage the data flowing from their simulations and into their post-processing systems. We envision that the scientists will be placing more in situ analysis and visualization techniques into their simulations, not by drastically modifying their simulations, but rather by using advanced I/O componentization frameworks such as ADIOS, so that their I/O pipelines can be managed. Once these techniques are incorporated into the code, we find that scientists do not want

possible bugs in the analysis and visualization codes to cripple or slow their simulation. They also need to discover ways to automate the in situ workflows, to allow for more dynamic execution of their expensive calculations, and to record and manage the provenance of their simulations in a resilient and power-efficient manner.

3 Service-oriented architecture

3.1 Next-generation I/O and file system

A key challenge for the exascale is the efficient organization and management of the large data volumes ingested and produced by scientific simulations, which are analyzed and visualized to support scientific investigations. The challenge exists (1) at the architecture or node level, to efficiently use increasingly deep memory hierarchies coupled with new memory properties like the persistence properties offered by NVRam or Memristors; (2) at the system level, to cope with I/O rates and volumes that by stressing the interconnect can severely limit application performance and/or consume unsustainable levels of power; and (3) at the exascale machine level, where immense aggregate I/O needs with potentially uneven loads are placed on underlying resources, resulting in data hotspots, interconnect congestion, and similar issues.

For future I/O systems at the exascale, we envision an environment at the node level in which multiple memory sockets, multiple coherence domains (i.e., "clusters" on a chip), multiple types of memory including persistent RAM, and disaggregation of memory with certain noncoherent memory are reachable via on-chip PCI or similar interconnects. In addition, caps on power consumption will limit power resources. This environment creates tradeoffs between data movement on-chip and on-node and the location of analysis and visualization computations on the data. We can move analysis code to where data currently resides, but doing so may introduce substantial variations in code execution times that can degrade the speedup of parallel codes. A more power-efficient solution may be one in which subsets of data are moved to other on-node memory, including persistent memory, before analysis is performed. In such cases, there exist both the necessity and opportunity to apply operations to data as it is being moved. Thus, even on-chip and on-node, tradeoffs will exist as to where and when analysis or visualization operations can or should be performed. For the I/O system, the overwhelming requirement is that there must be flexibility in (1) which operations are applied where on-node and (2) when they are applied, either synchronously with the application (potentially affecting its execution time) or asynchronously during on-node data movement.

Beyond individual nodes, operations that require use of the system interconnect are expensive in performance and power. As a result, analyses requiring more global data may preferably be carried out, in terms of both power and performance, by first moving data to some smaller number of nodes—in situ staging—and then analyzing it. Data staging also offers opportunities for hardware differentiation by endowing staging nodes with substantially more memory, with large persistent RAM, and with extensive flash-based storage. For I/O systems, this results in a need for multitier solutions for I/O, where analysis or visualization actions can be associated with I/O on-core, on-node, and in-staging. Analyses that occur in-staging can be substantially more complex than those on-node because of their inherently more asynchronous nature—that is, in-staging computations (especially for "fat memory" staging nodes) can operate on multiple output steps and will be bound by timing requirements determined by output frequencies.

The final memory tier for I/O is large-scale disk storage. We expect such storage to be attached to the exascale machine's periphery and/or to ancillary network-connected machines, and it will offer levels of throughput far less than that of the aggregate I/O actions of which the machine's cores are capable. As a result, for data analysis and visualization, a paramount requirement is to *bring code to data* and to do so dynamically, at the important output steps or simulation iterations and at specific cores and nodes.

3.2 SOA approach to I/O pipelines

The goal for SOA is to create a series of collaborating services that are published and available for invocation on the *service bus*. Most enterprises have adopted SOA to deliver the business agility and IT flexibility promised by web services. The benefits delivered by this approach come from its manner of coping with the increased complexity from the services themselves, from the service developers, and from the support and maintenance of each individual service. The SOA approach requires the creation of a service-oriented environment governed by three basic elements: (1) services can be published, discovered, and used in a technology-neutral standard form; (2) the SOA encompasses the policies, practices, and framework to ensure that the correct services are provided and consumed; and (3) at least two distinct processes are created for the provider and the consumer.

HPC changes the way we develop the SOA for the enterprise, in that it reduces the role of the discoverability of services over the WAN but places constraints on the performance and reliability of the services on the HPC platform. One example of using the SOA approach in HPC comes from our team's involvement with the CPES project. One of the goals of the project was to be able to both loosely and tightly couple multiple simulations. Each of the simulations, including the XGC code and the M3D code, is developed by an independent team and has dramatically different requirements on both the computational resources and the libraries used. These simulations can be coupled in two fundamental ways: (1) use a framework similar to MCT/ESMF, and compile the framework and codes together into a single executable, or (2) use the SOA approach, letting the I/O layer act as the communication mechanism for multiple executables to communicate with one another. The first approach has many advantages. For example, one controller code can enforce which simulation code executes on which processors, and MPI communication (which has been highly optimized on most HPC platforms) can be used between the codes. The advantages of the second approach are that each group can maintain the codes separately and we can easily switch from a file-based coupling to a memory-to-memory coupling if provided by the data transport method inside the I/O componentization.

In our previous work using ADIOS as the SOA interface layer, we developed a managed data staging transport, DataTap, that uses distributed predictive scheduling for perturbation avoidance. We also developed the Dataspaces method, which allows physics application services to interact at run time with each other and with services for data monitoring, data analysis and visualization, and data archiving. Dataspaces creates a semantically specialized, virtual shared-space abstraction that can be associatively accessed by all components and services in the application workflow.

In creating an SOA layer that can allow multiple components/services to interact efficiently with one another, several ingredients are essential. First, we use a semantically specialized, virtual shared space. This allows us to use a distributed set of nodes with a distributed hash table (DHT) that supports fast data-lookup operations as well as fast metadata propagation for data stores. Specifically, for multidimensional geometric domains,

we use a Hilbert space-filling curve (SFC) to generate the DHT index. Second, we allow for on-the-fly construction of a staging resource; and by using an RDMA layer such as DataTap, we provide asynchronous coordination and interaction between the services. Furthermore, we allow for complex, geometry-based queries and in-space transformations and manipulations of the data as it is being moved from one service to another.

By employing a SOA layer we can think about new paradigms for HPC to remotely debug and monitor the data. By using techniques such as ActiveSpaces and SmartTap for dynamic code deployment, we can provide programming support for data kernels to operate on data of interest. This allows us to dynamically deploy binary code to Dataspaces, execute it on the relevant data objects in parallel, and then return the results. Such ability is critical for SOA acceptance by the HPC community.

In the SOA approach, we allow plugins to be run statically on separate staging resources. As the number of services available to application teams continues to grow, workflow technologies must be deployed to schedule the services and the data movement from one service to another. Workflow technologies allow for defining a workflow of plugins, using limited resources to execute them, separating the definition given by the user from the execution framework, describing flexible resource requirements of plugins, dynamically starting and stopping plugins with a scheduler, and recording the provenance of actions and data lineage.

Previously, we worked with the Kepler workflow system for postprocessing scientific data and for real-time simulation monitoring on large-scale systems. Currently, we are creating a workflow system to work in situ that will be simplified compared to Kepler, since it needs to fulfill the requirements of high-performance computing—that is, it must use very little memory and be efficient in scheduling services with respect to resiliency and energy cost. The simplified workflows will form a directed acyclic graph based on the dependencies between their input and output data.

In summary, the essential ingredients needed for an SOA approach for HPC are to create a very lightweight interface that will look like an I/O interface and to schedule the data movement, binary kernels, and services to minimize energy cost and maximize productivity for the application teams. This realization must be easy to use and must default back to extremely reliable mechanisms for coupling the services. We have placed novel methods into the ADIOS framework, using DataTap, DataSpaces, SmartTap, and Activespaces to realize this vision. We have used this technology to couple the Pixie3D code to the Pixplot code for post-processing, Visit and ParaView for data visualization, and eSiMon for real-time monitoring. Once ADIOS was placed inside the code, no other changes were necessary for the codes to use either file-based coupling or memory-based coupling.

4 Next-generation frameworks for I/O pipelines

To take full advantage of the rapidly developing I/O technologies, a framework must provide a componentized architecture that can easily, without additional burden on the application developer, exploit the staging resources. This can be accomplished by a vertical partitioning of responsibilities in the framework stack. Thus, at the highest level the application developer should need to utilize only the data I/O APIs and metadata descriptions. Below this level, the framework should incorporate data management services that *enhance* the data output, such as data compression and indexing. Specific targeted optimizations for different architectures and data models should be encapsulated into a distinct layer of the I/O framework, allowing a multitude of applications and data

management services to work in concert with data transport. Additionally, the framework should not only utilize multiple data formats for output but also partition the complexity of optimizing each data format from the application developer. Finally, the framework should support the addition of schedulers who can manage the movement of data to reduce interference [10] and also manage the provisioning of resources to plugins in the staging area.

Of particular note is the addition of visualization to the stack, both as stand-alone applications that utilize framework APIs to access the data and as plugins that can be scheduled alongside other I/O pipeline operations, as shown in Figure 1.

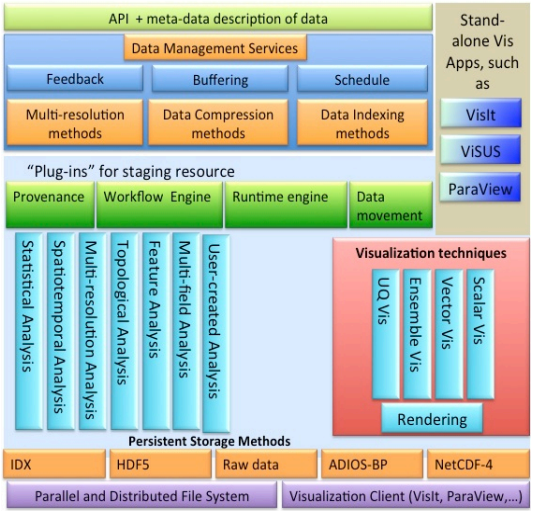

Figure 1: The I/O framework is partitioned vertically into distinct layers that improve both usability and manageability of I/O tasks.

Coupled with the goal of providing efficient data management through I/O pipelines, the framework must also be responsible for providing services for visualization and analysis, as well as preparatory tasks that aid the analysis operations such as sorting and binning [2]. Furthermore, to allow easy adoption, the application should not require significant modifications when switching between multiple data output and management services. By maintaining a partition between the data output API (which is part of the application source) and the control plane (which is responsible for describing the output data and selecting how it is output and what services are included in the pipeline), we can expose the users to only those parts of the framework that they require.

5 In situ data processing

The cost of data movement, both from the application to storage and from storage to analysis or visualization, is a deterrent to effective data use. The output costs increase the overall application running time and often force the user to reduce the total volume of data being produced by outputting data less frequently. Input costs, especially to visualization, can make up 80% [11, 12] of the total running time for scientific workflows, often increasing the time required to extract information from the simulation data. For many workflows, some of the operations can be performed without moving the data to and from storage; these operations are thus performed in situ or in place.

The advantage of in situ processing is that it allows the I/O pipeline to bypass a large portion of the input and output overheads associated with accessing the storage backend. The most common in situ processing technique is in situ visualization, where the data is prepared and visualized without I/O to disk. By holding the data on the node where it is generated and producing the visualization at the same node, the potential bottlenecks that arise from data movement are alleviated. This can be achieved without the use of an in situ data processing framework such as the one proposed here. However, tight integration of the visualization functionality into the application introduces a potential concern for resiliency, where errors in the visualization will cause application failures. With a service-oriented approach to visualization, the visualization operators can be functionally isolated from the application, thus allowing for resiliency.

Alongside in situ operations that require no data movement (i.e., the data is processed on the same node on which it is generated), a slightly looser form of this technique is operations that take advantage of the staging area. Instead of disk, the data is moved to the staging nodes, where additional processing is performed before output to disk. These in-transit operations see some of the benefits of in situ operations, such as reduced overhead from I/O, without suffering some of the problems, such as an increased duration for memory occupancy for the data. Such operations do require additional data movement compared to true in situ operations, possibly resulting in additional performance overhead. Because of the nature of asynchronous data movement, however, the perceived overhead for the application may actually be lower than for synchronous in situ operations. The development of an I/O framework for extreme-scale data processing should utilize a combination of both in situ and in-transit operations through the staging area.

6 Data staging

The complexities of the memory hierarchy described in Section 3.1 and the performance gains possible through proper utilization of this hierarchy have forced developers of I/O frameworks to strongly consider a shift in the paradigm of I/O processing. One such shift that has gained traction in the domain of scientific computing is the idea of data staging. Data staging extends the typical I/O system to include additional system nodes closely connected to the application nodes (see Figure 2). These "staging" nodes provide a transient buffer for the application to output data, allowing a decoupling of the application from the I/O resources. The user-level control on these nodes, in comparison with more traditional I/O nodes on many leadership-class computing systems, is an additional advantage allowing the development of processing plugins that utilize the available compute capabilities of these nodes. Thus, the staging architecture not only provides a high-performance data extraction mechanism but also allows the development of in-transit processing I/O pipelines. The design of the staging area is now being extended beyond just the allocation of entire physical nodes to encompass finer-grained resource partitioning, such as individual cores on the application nodes and GPUs.

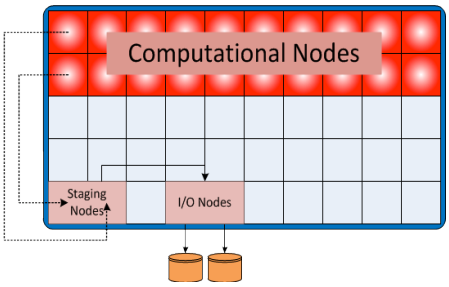

Figure 2: Staging area.

Our team's experience with data staging has also identified management and control of data movement as a major area of performance optimization. Without managing transfers, the movement of data to the staging area can interfere with intra-application communication, especially as we start scaling applications to very large numbers of cores. Introducing scheduling techniques into the control plane allows us to minimize this perturbation. Similarly, because of the constraints on resources available to the staging area, mechanisms are required to address the provisioning, allocation, and preemption of resources associated with staging area processing.

Once the data is extracted to the staging area, a variety of in-transit operations can be performed on the data, such as complex analysis, sorting, and *in situ* visualization. At Supercomputing 2010 we demonstrated the capability for *in situ* visualization, using the fusion application Pixie3D [7–9] to generate data that is visualized through ParaView [13] while also producing diagnostic information that is fed to eSiMon [14].

7 Data management

Data management challenges at the extreme scale require a rethinking of the underlying principle of data movement. As we approach exascale, it will become increasingly infeasible to output the entire data set to storage with no additional processing. Not only will the impact on the run time of the application be extensive, but a very large *input* overhead also will be incurred on the analysis and visualization components of the scientific workflow. Even today input costs dominate the performance of visualization systems, with some estimates placing them as high at 80% [11]. In order to efficiently handle the volume, data must be prepared before it hits the disk through techniques such as compression, multiresolution sampling, and indexing.

7.1 Data compression techniques

Given the growing volume of data that a storage system must handle, as well as such costs as moving these large volumes of data to the staging area, data compression provides a compelling case for *in situ* processing and I/O pipelines. However, scientific data has very high entropy, often resulting in small lossless compression ratios at the expense of additional computation. Inorder to address the specific concerns of scientific data, ISABELA [3] was introduced. ISABELA allows high compression of scientific data and also guarantees user-specified accuracy for the compressed data. First, the scientific data is linearized and divided into fixed-size chunks, each of which is called a compression *window*. Currently, ISABELA sets the window size at 256 data elements. This leads to a 2 kB window size for double-precision data. However, the window size is configurable. Within each window, the data is sorted based on their values by some permutation functions. Thus, for a window consisting of N data elements, each index value requires $\log_2 N$ bits, and the total storage for indices is $N * \log_2 N$ bits. Therefore, the vector length N is the only factor that determines the storage requirements for the indices. By sorting the data, the original highly irregular data becomes a monotonic curve, which can be more accurately modeled by using curve fitting. ISABELA applies B-spline to the sorted data values and records the coefficients. As any curve-fitting function inevitably introduces error, ISABELA additionally computes the relative errors between the actual data and the B-spline modeled data. Error values are also stored at compression time. Since the values within the windows are monotonically increasing, the vast majority of points have very small errors and are thus highly compressible. After the compression, the original data is no longer needed, and ISABELA stores three data structures: the per window B-spline coefficients; the permuted indices for each sorted window, each of size dlog(W)e bits; and the compressed quantized error values. These are then combined with an EQ-calibrator for error adjustment, allowing for a fourfold to fivefold data reduction. The accuracy of the reconstructed data is guaranteed to be within 0.1% point-by-point relative error, lower than 0.01 normalized RMSE, and higher than 0.99 Pearson correlation.

The availability of domain-aware compression in a service-based pipelined approach provides an interesting example of the challenges we will be facing. For example, it may be advantageous to perform data compression at the individual application nodes before data is moved into the pipeline, or it may be better to transmit uncompressed data from the application and perform data compression farther along in the pipeline. This decision will depend on factors such as the cost (both time and energy) of compression and decompression, cost of network data movement, and whether other operations will require uncompressed data.

7.2 Multiresolution technique

For very large regular meshes, the data can be organized by using techniques described in [15]. These multiresolution techniques describe the data at multiple levels of granularity, allowing the downstream processing tasks to select coarser granularities to reduce the data volume. One method of organizing data in a multiresolution stream is Z-indexing. For 3D block decompositions, for example, Z-indexing reduces the volume of data to $1/8^{th}$ its size with each decrease in the resolution.

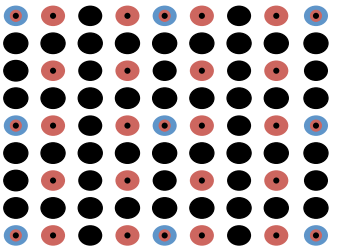

Figure3: Simple multiresolution format.

The use of multiresolution data formats is another technique we are exploring to reduce the data load on the pipeline. We envision there being certain tasks for which a full resolution dataset is unnecessary. By reorganizing outgoing data into a series of increasingly higher resolution subsets, a consumer of that data can choose to discontinue use of the data once it has reached a sufficiently high resolution for the task at hand. An example of this is a visualization task, where a large data set might be halted once a sufficient resolution is reached to identify a particular simulation feature or to determine that such a feature is not present.

Different approaches to multiresolution are possible depending on the requirements of the application. For instance, a simple approach is shown in Figure 3. Here, a 2D data set is shown, with its multiresolution decomposition indicated by the blue points (coarsest resolution), red points (medium granularity), and black points (finest resolution). We note that while reading a coarser resolution will exclude the finer resolutions, reading the finer resolution requires reading all resolutions with coarser granularities. Thus, reading the black points requires reading the entire data set, while reading the blue points requires reading only 1/9 of the data. Data from each resolution level is stored in the same order (e.g., column-major) as the full data set. This format allows data to be loaded progressively, beginning with a coarser view and becoming more refined as additional levels of resolution are loaded.

7.3 Data indexing and characteristics

As file sizes grow in complexity from the increased degrees of freedom and increased physics, efficient methods must be devised for querying data either “in-transit” or on the file system. These queries must be made efficient both in the energy they consume to produce the metadata necessary to quickly locate the data and in the total size of the data that it ultimately will consume on the file systems. These indices and characteristics can be stored and computed in multiple ways. The first is to compute simple statistics “communication-free” and then save them along with the data. As described previously, we store the data in chunks, and each chunk contains its own indices and characteristics. We start with a basic set, min/max/average/std. deviation, and allow the method builder to extend these without increasing the file size by more than 0.1% of the raw data.

Each data chunk can be compressed and stored in a multiresolution format, and each chunk can then be indexed and characterized. This approach allows complex queries to be performed on the basic index, and then the data is accessed if it meets the criteria of the query. One example comes from the GTS code, which is used in order to understand the turbulence intensity flux at each radial zone and their time correlations. This metadata can be stored by using FastBit [4] or ISABELA-QA plugins from the framework with no changes by the GTS scientist. By performing a query, the only data that is read in is the data that meets these criteria.

8 Monitoring

As simulation teams continue to grow, multiple scientists need to monitor their simulation when they run at scale. Scientists who will monitor a large-scale simulation come from different backgrounds. For example, in CPES we have performance and optimization experts monitoring simulations, along with I/O experts, computational physicists, and experimental and theoretical physicists. These scientists generally look for different pieces of the output and require different levels of simplicity in the technology. We developed the eSiMon collaborative portal [14] to monitor large-scale simulations from OLCF and NERSC and to provide a way for scientists to visualize their data during the run and to postprocess their data via a simple point-and-click mechanism. The eSiMon portal links users' activity on the front end to the corresponding raw simulation data using recorded data lineage information. By saving the provenance of the data, code, and environment, scientists can access information such as the performance of the code or compiler options used to build the code. Users can also perform built-in analysis on original data files or use fast data movers [16] for WAN data movement.

Previously, scientists had to run the Kepler workflow system in concert with their simulation; as data was written from the simulation to the file system, Kepler would direct new actors to read the data and process it to produce images and movies that could then be seen on eSiMon. Today, we have integrated eSiMon APIs into the ADIOS framework as a method. Scientists can now annotate their data to indicate to the system which variables will be visualized and how they are visualized. The data moves from the HPC machine to an external cluster to be transformed into a visualization, which can be shared during the simulation and displayed on eSiMon.

Since eSiMon provides the gateway for a diverse group of scientists to interact with their data, eSiMon must be able to attach to powerful analysis and visualization packages on the back end. It must also be able to submit jobs on HPC machines, move data and executables, and maintain the provenance from all the users. This enables many scientists to interact, collectively monitor their simulation, take notes, and share these with their collaborators. It also helps scientists maintain a repository of simulation runs, as well as associated inputs, notes, and visualizations.

A natural and desirable feature for the next-generation eSiMon should be to give scientists a "control" aspect of the run, including (1) starting and stopping the simulation; (2) maintaining a database of all the runs that a group submitted for the application, including all of the input parameters of the run when possible; and (3) providing the ability to get statistics for the input parameters for the runs (e.g., min/max of an input parameter). Such a system will ultimately allow us to educate new scientists, transferring expertise as our knowledge grows.

9 Usability for performance optimizations

Although new HPC technology is driven primarily by the performance requirements of application scientists, the adoption of new technologies is usually constrained by the lack of usability in extracting that same performance. For example, the penetration of the Cell architecture in HPC has been bogged down because of difficulties in writing applications optimized specifically for the architecture. Learning from this lesson, GPU technology is being promoted alongside infrastructure for more usable transformation of application kernels to GPU code using technologies such as CUDA [17,18] or OpenCL [19].

Similarly, for I/O an important challenge facing scientists today is to drive adoption of new technological innovations in the space by real application users. A failure in the usability of I/O optimizations in many I/O libraries has caused actual I/O performance to be severely deficient compared with potential peak throughput. Further optimizations in I/O are often targeted to specific architectures and may not translate to other systems, limiting portability and further constraining usage.

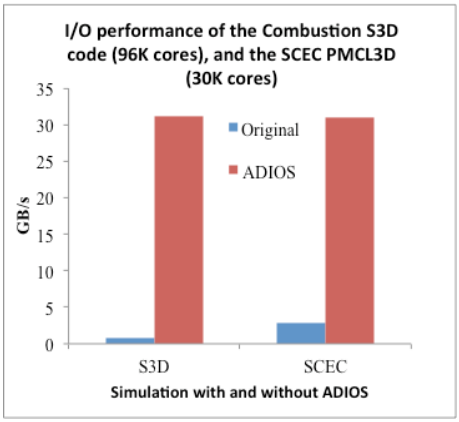

Figure 0: ADIOS methods show high performance across multiple applications.

The next generation of proposed I/O frameworks must address this concern by clearly partitioning the task of optimizations for I/O from the actual description of the I/O. The latter is the responsibility of the application developer alone, while the former can be delegated to specialists for each of the systems concerned. Additionally, the selection of the I/O techniques should be deferred at least to job submission time or even later to run time. This approach allows the application to be optimized across multiple platforms and architectures, and similarly the optimization can be used for multiple applications. In this section, we describe a sample optimization developed by our team for significant improvements to write performance on the Cray XT5 Jaguar at ORNL.

Achieving write performance requires consideration of many factors. These include the number of writers and distribution of data in memory, type and configuration of file system, number and performance characteristics of disks, interconnect bandwidth, and current utilization of available I/O resources.

One technique that has been developed by our team utilizes write aggregations. Two levels of write aggregations are used. At the first level, the scheme aggregates local process data in a procedure known as *write behind*, allowing smaller writes to be amalgamated into more optimal-sized write requests. At the second level, processors act as aggregators across subsections or groups of the entire parallel application. Once the aggregation is under way, the data is output to a *subfile* that serves as a stripe on a single storage target to minimize write-based lock contention in the file system. In order to maintain a global view of the data, a corresponding metadata file is also written out.

The complexity of this technique, as well as the specific parameters that depend on the characteristics of the execution platform (in this case, the Cray XT5 Jaguar), makes it unlikely to be directly incorporated into more than a handful of leadership-class computing applications. However, because of the support provided by the ADIOS framework for pluggable methods and run-time selection of output methods, any application using ADIOS for I/O, as shown in Figure 4, can easily apply this method, resulting in substantial performance gains without requiring any additional effort by the application developer.

9.1 Next-generation file formats using an elastic data organization

As file systems increase in complexity due to high levels of concurrency, we need to understand how to place data on the file system and how to decrease the overhead of both writing and reading data to and from these complex file systems. The file system at the OLCF, for example, is shared among many machines and provides over 670 object storage targets (OSTs), where each OST has a RAID set associated with it. To gain performance in writing and reading, we have found that data must be “chunked” and that each chunk must be placed on individual storage targets. Furthermore, we need to manage/schedule the communication of data

from the individual processors to the OSTs, rather than just having the file system take care of the writing. We have found [20] that by carefully managing and scheduling these, and using techniques such as subfiles, we can attain low overhead synchronous writes. For example, a recent S3D [21] calculation running on 96K cores, writing 1.9 MB of data per core, can take 0.6% of the wall-clock time in I/O. Recent advances in HDF5 [22] also show dramatic increases in I/O performance by chunking the data, rather than by writing logically contiguous files.

File formats should be easy to use, easy to program, high performance, portable, self-describing, and able to work efficiently on parallel file systems and simple file systems. They should also allow for simple queries into the data, as described in this paper. A major challenge has always been in the understanding of the performance of I/O for both reading and writing, and in trying to optimize for both of these cases.

When we systematically examined the read performance of scientific applications such as S3D on large-scale supercomputers, we saw a huge disparity of performance from many of the common access patterns that scientists use to read data [23]. The six most common access read patterns are as follows: (1) all data is written and read from the same or a different number of processors, (2) all of one variable is read from a complete output set, (3) all of a few variables using any number of processors are read, (4) a plane in any dimension of data is read for postprocessing, (5) an arbitrary rectangular subset is read in from the multidimensional data set on an arbitrary number of processors, and (6) an arbitrary area on an orthogonal plane is read in on an arbitrary number of processors. Much to our surprise, file formats that chunk the data routinely achieved better read performance than logically contiguous file formats.

To understand why chunking of data improves read performance for most of the common read patterns that occur in data analysis, we can look at a simple example faced by many when accessing a multidimensional variable. The first explanation of performance is in the disparity between the order of storage and the order of access for data elements in a multidimensional variable. Second, there is a lack of data *concurrency* when a subset of data elements is retrieved from a multidimensional variable.

Using a simple formulation to understand the concurrency, we found that when the subset of data elements is not logically contiguous, it is often concentrated on only a few storage devices among a large number of total devices that are used to store the entire variable, if the striping parameter is not configured in a sophisticated manner.

We have extended the ADIOS-BP file format with an elastic data organization (EDO) that can support flexible data organization strategies for different scientific applications. EDO provides two levels of data ordering algorithms. At the top level, it uses Hilbert space-filling curves (SFCs) to balance the distribution of data groups, referred to in ADIOS as *process groups*, across storage targets. SFC distributes data elements across parallel storage targets to aggregate their bandwidth without file system restrictions and improve concurrency for the planar reads. At a low level, EDO divides data elements within a data group into *subchunks*, and it balances the cost of seeking through *subchunks* and that of reading data from them. Figure 5 shows the performance of reading S3D data written from 4,096 cores using the ADIOS-BP file format

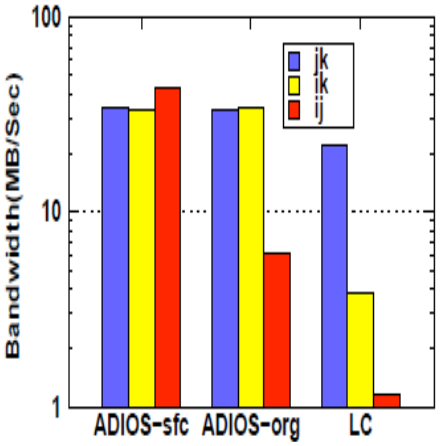

Figure 5: Performance improvements from elastic data organization in ADIOS.

(ADIOS-org), compared with using the EDO (ADIOS-sfc) and a conventional file format that organizes the data with logically contiguous MPI-IO. We then look at three different 2D planes and the I/O performance. Clearly, in this case EDO gives a higher and more consistent level of performance compared with other types of file formats.

10 Conclusions

The challenges of extreme-scale scientific data processing cannot be efficiently handled by using conventional techniques for I/O processing using store/retrieve/process operations that are bottlenecked by conventional storage technologies. In this paper we have proposed a shift in the paradigm for scientific data processing relying on emerging techniques in data staging, in situ data processing, data compression, multi-resolution data streams, fast bitmap indexing, and user accessible application monitoring. These techniques, utilized through the already established service-oriented approach to building pipelines and elastic data formats to encompass a variety of user needs, form the conceptual underpinnings of a next-generation I/O framework that can alleviate many of the concerns with I/O as leadership systems begin to scale toward exascale. We have also utilized our experience with building a successful I/O stack, ADIOS, to place manageability and usability of the framework as a first-class design goal. The goals of the framework described in this paper should serve as guidance for the evolution of the extreme-scale data processing domain.

References

[1] J. Bent et al., "PLFS: a checkpoint filesystem for parallel applications," in *SC*, 2009.

[2] F. Zheng et al., "PreDatA - Preparatory Data Analytics on Peta-Scale Machines," in *Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2010)*, 2010.

[3] S. Lakshminarasimhan et al., "Compressing the Incompressible with ISABELA in situ Reduction of Spatio-Temporal Data," presented at the 17th International European Conference on Parallel and Distributed Computing (Euro-Par 2011), 2011.

[4] K. Wu et al., "FastBit: Interactively Searching Massive Data," *J. of Physics: Conference Series,* vol. 180, 2009.

[5] R. Barreto et al., "Collaboration Portal for Petascale Simulations," *Proceedings of the 2009 International*

[6] J. Lofstead, et al., "Adaptable, Metadata Rich IO Methods for Portable High Performance IO," *IEEE International Symposium on Parallel & Distributed Processing,. IPDPS 2009,* pp.1-10, May 23-29, 2009.

[7] L. Chacón, "An Optimal, Parallel, Fully Implicit Newton-Krylov Solver for Three-Dimensional Visco-Resistive Magnetohydrodynamics," *Phys. Plasmas,* vol. 15, 2008.

[8] L. Chacón, "Scalable solvers for 3D magnetohydrodynamics," *Journal of Physics Conference Series,* vol. 125, 2008.

[9] L. Chacón, "A Non-staggered, Conservative, div(B)=0, Finite-Volume Scheme for 3D Implicit Extended Magnetohydrodynamics in Survilinear Geometries," *Comput. Phys. Comm.,* vol. 163, pp. 143-171, 2004.

[10] H. Abbasi et al., "DataStager: scalable data staging services for petascale applications," in *HPDC '09*, ed. Garching, Germany: ACM, pp. 39-48, 2009.

[11] T. Peterka et al., " End-to-End Study of Parallel Volume Rendering on the IBM Blue Gene/P," in *International Conference on Parallel Processing, 2009,* pp. 566-573, Sept. 22-25, 2009

[12] H. Childs et al., "Extreme Scaling of Production Visualization Software on Diverse Architectures," *Computer Graphics and Applications,* 2011.

[13] C. C. Law et al., "An Application Architecture for Large Data Visualization," in *PVG 2001: Proceedings of the IEEE 2001 Symposium on Parallel and Large-data Visualization*, 2001.

[14] R. Barreto et al., "Collaboration portal for petascale simulations," in *Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems,* pp. 384-393, 2009.

[15] V. Pascucci and R. J. Frank, "Hierarchical Indexing for Out-of-Core Access to Multi-Resolution Data," in *Hierarchical and Geometrical Methods in Scientific Visualization*, pp. 225-241, 2002.

[16] Scientific Data Management Research Group. (2009). *SRM-Lite*. Available: https://sdm.lbl.gov/twiki/bin/view/Software/SRMLite

[17] NVIDIA Corporation, *What is CUDA?* 2011.

[18] J. Nickolls et al., "Scalable parallel programming with CUDA," *Queue,* vol. 6, pp. 40-53, 2008.

[19] Khronos Group, *OpenCL - The Open Standard for Parallel Programming of Heterogeneous Systems,* 2011.
[20] J. Lofstead et al., "Managing Variability in the IO Performance of Petascale Storage Systems," in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC'10, 2010.
[21] J. H. Chen et al., "Terascale Direct Numerical Simulations of Turbulent Combustion Using S3D," *Computational Science & Discovery,* vol. 2, p. 015001, 2009.
[22] M. Howison et al., "H5hut: A High-Performance I/O Library for Particle-Based Simulations," in *Proceedings of 2010 Workshop on Interfaces and Abstractions for Scientific Data Storage (IASDS10)*, Heraklion, Crete, Greece, 2010.
[23] J. Lofstead et al., "Six Degrees of Scientific Data: Reading Patterns for Extreme Scale Science IO," in *The 20th International ACM Symposium on High-Performance Parallel and Distributed Computing*, San Jose, 2011.