Extending I/O through High Performance Data Services

Hasan Abbasi [#1], Jay Lofstead [#2], Fang Zheng [#3], Scott Klasky [*4], Karsten Schwan [#5], Matthew Wolf [#*6]

[#] *College of Computing, Georgia Institute of Technology*
Atlanta, GA, 30332
[1] habbasi@cc.gatech.edu
[2] lofstead@cc.gatech.edu
[3] fhzeng@cc.gatech.edu
[5] schwan@cc.gatech.edu
[6] mwolf@cc.gatech.edu

[*] *Oak Ridge National Laboratory*
Oakridge, TN, 37831
[4] klasky@ornl.gov

***Abstract*—The complexity of HPC systems has increased the burden on the developer as applications scale to hundreds of thousands of processing cores. Moreover, additional efforts are required to achieve acceptable I/O performance, where it is important how I/O is performed, which resources are used, and where I/O functionality is deployed. Specifically, by scheduling I/O data movement and by effectively placing operators affecting data volumes or information about the data, tremendous gains can be achieved both in the performance of simulation output and in the usability of output data.**

Previous studies have shown the value of using asynchronous I/O, of employing a staging area, and of performing select operations on data before it is written to disk. Leveraging such insights, this paper develops and experiments with higher level I/O abstractions, termed "data services", that manage output data from 'source to sink': where/when it is captured, transported towards storage, and filtered or manipulated by service functions to improve its information content. Useful services include data reduction, data indexing, and those that manage how I/O is performed, i.e., the control aspects of data movement. Our data services implementation distinguishes control aspects – the control plane – from data movement – the data plane, so that both may be changed separably. This results in runtime flexibility not only in which services to employ, but also in where to deploy them and how they use I/O resources. The outcome is consistently high levels of I/O performance at large scale, without requiring application change.

I. Introduction

Scalability is a common challenge for the developers of modern HPC applications, requiring careful code and performance tuning for petascale compute tasks. With data sizes increasing commensurately, data output is becoming an almost equally challenging task, where the conventional practice of storing data on disk, moving it off-site, reading it into a workflow, and analyzing it to produce scientific solutions becomes harder due to large data volumes and limited backend speeds. In fact, even the time required to move data to storage can become an obstacle, since if output actions cause an application to block on I/O, countless numbers of compute cores sit idle waiting on output. In addition, science is affected if it takes say, 500 seconds to output data, since that makes it hard to justify writing data more than once per hour, even if the science may benefit from more frequent output.

We are exploring alternative ways of performing output. Our approach goes beyond simply accelerating output to also moving select data manipulation tasks traditionally placed in an offline workflow 'into' the fast data path on the supercomputer. Suitable tasks include those that reduce data without loss of scientific validity, generate metadata (e.g., indexing) for easier data access, or perform lightweight analysis tasks for online validation of application 'health' via dashboards. While previous work has demonstrated the utility of this approach [35], [1], [4], [16], what remains lacking are system abstractions and the underlying runtime support for efficient I/O task representation, deployment, execution, and management. This paper presents such abstractions, termed 'data services', and uses them for high performance output on petascale machines.

Data services are system-level abstractions that encapsulate methods for data movement and manipulation on the fast path of data output. Data services are created and deployed separately from application codes, thereby separating changes made to application codes by science users from changes made to I/O actions by developers or administrators. Data services can be run asynchronously from the large-scale simulations running on supercomputers, in order to decouple I/O actions and their performance behavior from the those of the computations

performed by large-scale applications. Thus, science end users can focus on their codes, and administrators or developers can help create and manage I/O processes.

To make I/O processes manageable, data services are associated with I/O so as to retain flexibility (1) in the resources they consume, (2) in where services are run (e.g., on compute nodes and/or on staging nodes), and (3) in how and when they are run, including explicitly scheduling their execution to avoid perturbing the petascale simulation [2]. Flexibility in the levels of resources dedicated to I/O ranges from 'none', where compute nodes are used to run output actions, to cases in which a substantial number of 'fat nodes' are used for staging data and manipulating it prior to storing it on disk. Flexibility in service placement includes placing certain data reduction actions close to the source, even directly on compute nodes via the *SmartTap* abstraction used for capturing output data. Data produced by SmartTaps can be fed to storage and/or to additional data services placed on staging nodes, where data may be buffered, annotated, or reorganized prior to moving it to storage. Flexibility in how services are run is supported by output scheduling methods that take into account supercomputer interconnects with respect to perturbations of simulations caused by output activities, and by backend-sensitive methods that adapt to different storage characteristics and behaviors. Examples of the latter are those that avoid using storage targets already used by other parties or more simply, that use the appropriate number of storage targets to maximize throughput.

The data service abstraction makes it easier to develop, deploy, and maintain per-site and per-run optimizations of I/O processes. The abstraction also exploits the facts that there exist many non-trivial I/O and data processing tasks that can be done with few additional computational resources, on compute and/or on staging nodes, and moreover, that performing such tasks can result in performance gains when writing data and/or in usability gains by increasing the information content of data through annotation. Toward these ends, data services are defined to differentiate between (1) data extraction, using the SmartTap abstraction, (2) data processing via lightweight computations associated with data services, and (3) data storage using methods that take into account storage system characteristics and behaviors, and (4) for flexibility, the implementation of data services separates data movement and manipulation – the data plane – from how such actions are managed – the control plane. Therefore, new scheduling methods for data extraction or new techniques for how storage targets (or other backends) are used can be deployed easily, without changing data plane movement and processing actions. The importance of this flexibility is demonstrated in this paper with performance results on how and in which fashion data is evicted to the storage system.

Data services are evaluated with representative petascale codes running on the Cray 'Jaguar' leadership machine at Oak Ridge National labs. One explicit example is the Gyrokinetic Toroidal Code (GTC)[33], which is a 3-dimensional particle-in-cell code used to study micro-turbulence in magnetic confinement fusion from first principles plasma theory. As more complex phenomenon are explored, the total number of ions being modelled increases to trillions of total ions, resulting in data rates of up to 100 TB/run. The data produced by GTC is stored and processed through workflows producing scientific understanding. As the size of the data generated increases, however, it becomes increasingly difficult to justify the time cost of storing all data on disk and then eventually, reading it back into a workflow module. A better use of resources is one that employs data services to first move data to a temporary, non-disk staging area and if needed, process it 'in-transit' to make it more amenable for later analysis and/or to quickly extract scientific insights from the GTC code's execution. Services applied to GTC output data include data reduction and format conversion. Given the enormous data volumes involved, however, this paper demonstrates that the successful use of such data services also requires the careful management of how data is extracted via SmartTaps, how services are applied in the staging area, and how data is moved to disk. For a second petascale code with smaller volumes of output data, the CHIMERA[23] multi-dimensional supernova simulation, a more complex service indexing output data is shown feasible, the intent being to improve the ease with which workflows can later access and use output data for detailed analysis.

Performance results attained with the GTC and CHIMERA codes on Jaguar and Jaguarpf demonstrate the power of the data services approach. Using CHIMERA we observed that the performance impact of data extraction at 8192 cores decreased from 37.84% with pHDF5 to only 0.14% with the Datatap, while using only 0.78% additional resources. At 65k cores our evaluation of GTC restart output demonstrated a reduction in overhead from 42.5% to only 10.9%. Moreover, evaluation of the data format conversion service for CHIMERA showed that decoupling the processing of the output from the application had a minimal impact on application performance.

The remainder of this paper is structured as follows. In Section II we discuss other research efforts similar to data services and elaborate on the previous work that laid the foundations for this paper. Section III elaborates on the data service abstraction and describes the salient features of our implementation. We evaluate the performance benefits of the data service architecture in Section IV using CHIMERA and GTC as example applications. We also evaluate the importance of managing

data output to disk in Section IV-D. We conclude with a description of the contributions of this paper and describe our future research direction we plan in Section V.

II. Related Work

I/O performance is a significant topic of research with many efforts at studying scalable implementations for file systems. Filesystems like PVFS[18], [7], Lustre [8] and GPFS [28] seek to provide a high performance platform for I/O, offering scalability to hundreds of thousands of clients. Due to the general applicability and the traditional filesystem semantics for data and metadata consistency, there is a high level of user burden when optimization applications for these platforms. Given the requirements of application portability, the task of optimization is complicated even further, requiring fine grained tuning in order to achieve the best performance. This coupling of the storage platform and the application results in sub-optimal performance characteristics when the underlying infrastructure changes. The data service abstraction does not replace the traditional filesystem but insulates the application from the filesystem allowing for independent evolution of the control infrastructure. This decoupling of the control plane increases the applicability of optimization operations.

LWFS is a lightweight filesystem that resolves some of the performance bottlenecks [24], [25] that closely relates to our work on asynchronous lightweight data movement for storage processing. LWFS is analogous to the data extraction service but does not address the processing of in-flight data in order to manage the overheads. The management infrastructure described here can work in concert with the LWFS architecture and data can be post-processed after storage as described in [34].

PDIO[31] and DART[11] provide a infrastructure similar to the Datatap in order to move data from the compute nodes. The data service abstraction can be used for both platforms and the functionality and performance benefits of managing the data movement and in-line processing can be realized for either system.

Past efforts into studying Active Disks[27], [26] addresses the same domain as data services. Active Disks rely on data operations hosted near the storage target, thus limiting the scope of problems that can be addressed in this manner. In contrast the data service abstraction is centered around a flexible approach to the location of both data consumption targets (of which disks are only one example) and data processing operations. This level of flexibility also enables data services to utilize active disk style functionality within the architecture.

Workflow systems such as Kepler[22], Pegasus [10] amongst others [36], [32] approach the important task of data processing as an offline task. One limitations of such approaches is the reliance on stored data as input for the processing pipeline. The advantage is that workflow systems can be used for more complex operations than a data service due to the significantly larger buffering space provided by on-disk storage. Such systems can still be part of the data service pipeline as post-storage actions [34].

Map-Reduce [9] from Google is an alternate abstraction for data processing that has gained popularity in the last few years. The Map-Reduce abstraction also provides an alternate view into massive scale data processing. However, the focus for Map-Reduce is on processing data from local storage instead of dealing with streaming data generated from a running application. Many typical data services, such as data reductions, can be cast into the Map-Reduce framework and the programming model can be used with some modifications for the development of data services.

Our previous research on publish subscribe systems [14] forms the foundation of our motivation for data service. The evolution of ECho, EVPath [12], is used as the data transport for the staging area as well providing a mechanism to create dynamic overlays for data processing.

The Data Workspace described in LIVE [1] is the immediate predecessor of the data service. LIVE did not address the issues of scalability and the requisite management of data processing and movement.

ADIOS [21] is a high performance API for componentizing the I/O stack. It allows the scientific coder to place a generic component call into the code, and then the end user can at invocation time configure the particular optimized component that is appropriate to the current run-time conditions. This gives the end user much more control, but also leaves much of the onus on the end user to make decisions. This work extends the ADIOS interface by providing more autonomic and richer data services through the same componentized interface.

III. The Data Service Abstraction

A *data service* is a collection of actions on 'in transit' data, carrying out tasks like data extraction, data staging, formatting, indexing, compression, and storage. A high level depiction of the Data Service architecture appears in Figure 1, which shows that conceptually, running a Data Service has four distinct phases:

1) from bytes to structured data: writing and formatting output into buffers to create data items of well-defined structure and memory layout;
2) controlling data movement: extracting data items to maximize application performance;
3) online data processing: applying service codes to output data while it is being moved; and
4) flexible data consumption: moving data out of the petascale machine (e.g., to storage or the network) sensitive to backend characteristics and behaviors.

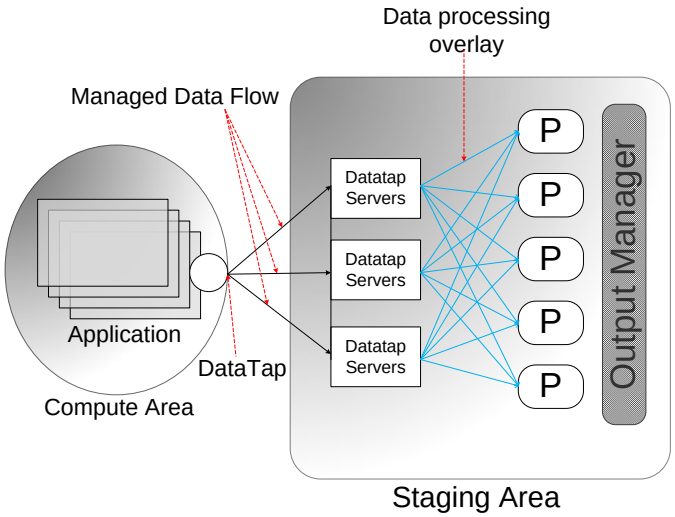


Fig. 1. High level view of the Data Service Architecture

In order to support customizable and configurable I/O data movement and processing, each of these steps are carried out in ways that are defined by control methods associated with them. Examples include the scheduling of data eviction from compute nodes and the controlled movement of data to storage subsystems. Each of these steps and their controls are explained next.

A. *SmartTap: Structured Output Data*

With the massive quantities of data generated by petascale applications, it is not uncommon that only a fraction of this data is actually required for scientific analysis. Thus, producing and outputting the entire data set and then later extracting a smaller portion for post-processing can create an unnecessary performance bottleneck. However, identifying "useful" data is highly specific to each scientific undertaking, requiring in-depth knowledge from the user as well as application hints that enable this reduction. For example, for a molecular dynamics application, a user may only be interested in the characteristics of particles in a small bounding box. Traditionally, this requires the application to provide the functionality that allows output in a bounded space, thus making it necessary for end users to change application code. In contrast, data services allow end users to flexibly associate such functionality with the data output process itself. This begins with the data capture and formatting steps performed by the *SmartTap* client module running on compute nodes.

The first task of SmartTap is to deal with a necessary prerequisite for applying operations to output data, which is to represent such data in ways amenable for processing. Concretely, this means that data must have some well-defined structure or type and that services are aware of these types. For Data Services, such type information is first specified by end users when output data is created, via the ADIOS [21] API now used by a variety of petascale applications. Such specifications, then, are used by SmartTap to create in memory buffers containing structured data, using an efficient self-describing binary data format, termed FFS [13], [15]. The SmartTap implementation does so with high performance and low overhead, by exploiting the dynamic binary code generation capabilities provided by the COD package [3], [5]. Since FFS formats are self-describing, with FFS and COD, data services can be written and used without a-priori knowledge of data structure and memory layout. This makes it easy to write new services and dynamically deploy them 'into' the data output pipeline wherever and whenever needed, potentially even while I/O is ongoing [30]. We note that we do not propose FFS as a new standard binary data format, but simply offer it as an efficient and compact, intermediate binary format used by data services. Developers use FFS when implementing data services, but these formats are converted into standard forms for end users, e.g., when storing data on disk using the HDF-5 data format.

Datatap internally uses FFS to encode data at client side so that the data received by Datatap server contains structure information which includes variable name, data type, array rank/bound/offsets, etc. The overhead of encoding data with FFS is negligible (cite some PBIO paper). Datatap Servers can then use those information when writing data to certain format. For example, Datatap servers use those structure information to create dataspace and hyperslabs when writing data to HDF5 file.

Some optimizations can be applied at Datatap servers to speedup data formatting. One is that the data chunks received from multiple compute nodes can be combined to large sequential chunks to improve the performance of writing data to file. The other optimization is that mul-

tiple Datatap servers can use parallel I/O for writing if the the format support parallel I/O access. For example, multiple Datatap servers can use pHDF5 API or ADIOS-MPI method to write data to a single shared HDF5 file or BP file.

A second task of SmartTap is to move output data to downstream data services, for additional processing, for storage, or both. SmartTap achieves this by asynchronous use of RDMA, which implies that data movement is under control of a SmartTap server, termed Datatap server in Figure1. The advantages of this approach are elaborated in [2], and additional discussions of server-directed I/O for managing data movement appear in [29], [17], [25]. Briefly, one reason for using server-side I/O in petascale machines is that servers can address the inevitable impedance mismatch between differently sized compute vs. I/O partitions, using per-application customized solutions for data movement and manipulation. This also takes advantage of the fact that petascale machines are typically configured with many compute nodes vs. a smaller number of 'fat' nodes, the latter being well-suited for running data services.

A final, optional task for SmartTaps is data annotation, where richer meta-data can facilitate downstream data analysis and help scientists gain insights into massive datasets. Specifically, when output data is formatted on compute nodes and again, when it is moved to SmartTap servers, this provides opportunities to generate interesting meta-data from raw output data and even obtain select aggregated global characteristics. Here, the role of the SmartTap is to generate per compute node meta-data, whereas the server carries out aggregation or summarizing actions. Useful and feasible operations jointly performed in this manner include histograms, bitmap indices, PDF, linear regression, etc. For these, SmartTaps generate meta-data like min/max/sum values (such lightweight client-side calculations have negligible overhead [19], which are then attached as 'attributes' to the data fetch requests sent to the server. The server reads these attributes into temporary internal state and interacts with other servers to calculate desired global values.

B. Controlled Data Movement

The task of extracting data into an output pipeline can have significant overhead as data sizes increase. Asynchronous data extraction can exhibit performance penalties from perturbation on the interconnect shared by the application. Unmanaged data output to disk is seen to have high performance penalties due to inappropriate use of the shared storage hardware. In response, SmartTap servers not only execute functions that move data – in the data plane, but they also control how data is moved – in the control plane. Examples of the latter are the scheduling methods for data extraction described in [2], which extract data from compute nodes in ways tolerant of and sensitive to how the application produces data and how it uses the petascale machine's interconnect. The twin goals met by such scheduling are to minimize application perturbation from interference due to blocking and shared use of the interconnect while also maximizing the throughput of data output. Results attained in our earlier work show the extreme scale at which such interference effects occur, starting with 1000+ node applications and demonstrated for up to 10,000+ nodes. Results shown in this paper demonstrate the equally important aspect of controlling and customizing how data is evicted from the staging area to disk (see SectionIV-D). For the data services control plane, this means that it must enable the use of alternative and potentially, application-specific strategies both for scheduling (i) the data extraction process and (ii) the way in which data is presented to the storage backend (which in the case of Jaguar, is the Lustre parallel file system).

C. Online Data Processing

As data moves from the generation point to the eventual data consumer (e.g., to disk storage or to an online data visualization), there is both a necessity and an opportunity for in-transit processing. Necessary processing services include those that convert data into the standard forms required by backend, such as the HDF-5 or NetCDF formats used in file systems. In our earlier work [20], we observe that for some workloads, it may cause unacceptable overhead to writing data into HDF-5 format synchronously due to synchronization and consistency maintenance. Performing data formatting in data services 'offloaded' to the staging area can help reduce these overheads. Other examples of data services are those that seize opportunities for performance improvements through data reduction, improve the accessibility of output data through data indexing, or perform tasks meaningful to applications like generating histograms or validating output data.

The data services infrastructure supports the 'in-transit' processing actions described above, but it does not provide the services themselves. This is in recognition of the application-specific nature of most services. The base support provided includes (i) the aforementioned formatting into structured data instead of using raw byte streams, coupled with the runtime provision of such type information to those services that need it, (ii) the ability to efficiently associate meta-data with output, as 'attributes' that can 'tag' typed data items, and (iii) rapid inspection of attributes to guide service actions. (iv) Data services themselves are implemented as dynamically linked libraries or as COD-generated codes placed into the SmartTap server's address space or into additional processes running on staging nodes. Simple examples are services run in SmartTap servers that calculate meta-data by first gathering all data fetch

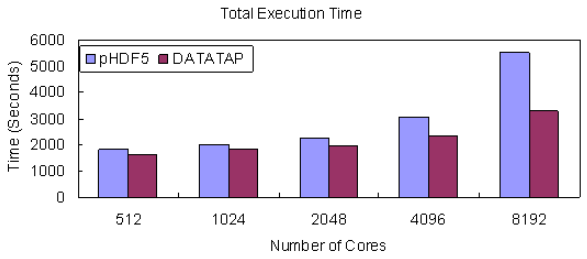
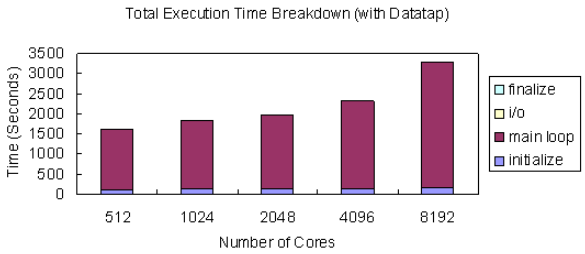
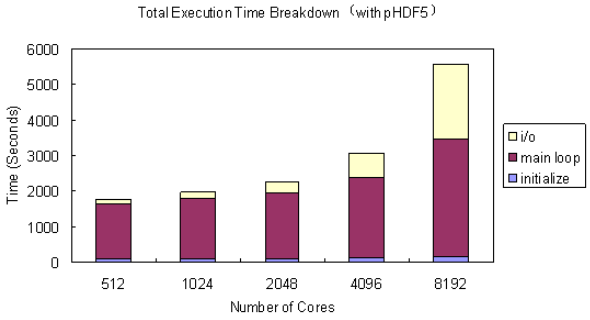


Fig. 2. Total Execution Time for CHIMERA



(a) Total Execution Time Breakdown with Datatap



(b) Total Execution Time Breakdown with pHDF5

Fig. 3. Comparisons of execution time for CHIMERA

requests from SmartTaps, then extracting the SmartTap-computed partial values represented as 'attributes' associated with such requests, and finally and as needed, exchanging information with other servers to obtain global values (e.g., min/max/sum values). More complex services may be realized as multiple processes forming an I/O pipeline, an example being the histogramming service for the CHIMERA code described in SectionIV-C.

D. Flexible Data Consumption

Once processing is completed, data must be evicted to backends like the disk, the network, or online data visualizations (e.g., to the dashboard created for supervising the GTC application). Using staging area resources, this may involve creating the 'right' number of disk output processes (e.g., to match the number of object stores used by the file system), it may involve the use of additional processes on staging area codes for data preparation for visual display on dashboards, and/or it may require careful admission control and message scheduling for network backends[6]. With the data services architecture, this involves creating, using, and connecting into an I/O pipeline any number of data services placed into any number of processes running on staging nodes. This is done with the EVpath middleware [12] that underlies data services and offers facilities for runtime service creation, deployment (incl. connection), configuration, and re-configuration[30].

IV. Experimental Evaluation

A. Example Application Scenarios

1) CHIMERA: CHIMERA[23] is a multi-dimensional radiation hydrodynamics code designed to study core-collapse supernovae. We look at the periodic restart output that is used for both checkpointing and post-processing. The restart data consists of 80 scalar and arrays. Global arrays are regularly distributed among a 2-D grid of MPI processes. In our experiments each MPI process writes out approximately 930KB of data in each I/O phase.CHIMERA uses the ADIOS API [21] for I/O allowing multiple methods to be compared by simply modifying a variable in the configuration file. The data is defined as part of an external XML configuration with both structure and meta information and enabling the use of structured FFS data for output purposes. We have instrumented the application with specific calls to ADIOS in order to provide phase information to the underlying transport method, allowing us to customize the behavior of the data transport.

We evaluated two aspects of the data service for CHIMERA, viz. the extraction of data to the staging area and the conversion of the intermediate FFS format data to the HDF-5 format. Evaluation was performed at application sizes ranging from 512 to 8192 cores. 5 test runs are done at each size, with additional compute nodes serving as the data staging area. In all experiments with the Datatap we kept the ratio of compute nodes to staging nodes at 512 to 1. The application ran for 400 iterations and a restart output was done every 50 iterations.

The CHIMERA evaluations were performed on the NCCS Cray XT4 Jaguar at Oak Ridge National Labs. Each Jaguar node is a single socket, quad-core AMD Opteron running at 2.1 GHz with 8 GB of memory (2 GB/core). The network interconnect is the Cray Seastar2 with low level access provided through the Portals API. The compute nodes operating system is Compute Node Linux (CNL).

2) GTC: Gyrokinetic Toroidal Code (GTC) [33] is a 3-dimensional particle-in-cell code used to study micro-turbulence in magnetic confinement fusion from first principles plasma theory. GTC is highly scalable and we have performed evaluations on sizes from 16k processing elements to 112k processing elements.

In order to study the largest I/O element we have only studied the performance implications of the GTC restart output. The GTC restart output is approximately 10% of the overall problem sizes and provides a look at the extremes of I/O performance. Similar to CHIMERA, we use the ADIOS [21] library to perform I/O, providing us with a unique opportunity to study the implications of different methods for data extraction without modifying the application code. Restart was performed every 10 iterations, and the application was run for a total of 100 timesteps. The performance of data extraction service is compared to a special ADIOS method, NULL. The NULL method does not perform any data output and provides the base case for application runtime.

As described in [2] we instrumented GTC with programmatic hints to inform the data service controller about transitions to a compute phase.GTC evaluations were performed on the NCCS Cray XT5 Jaguarpf. Each node is configured with two quad-core AMD Opterons at 2.6 GHz with 16 GB of memory (2GB/core). Like the Cray XT4 used for CHIMERA, the Cray XT5 also uses the SeaStar2+ network interconnect programmed through the Portals API.

As an example of managed data extraction we used the PA_Con_1 scheduler described in [2]. The scheduler uses phase knowledge from the instrumented application in order to reduce potential interference with intra-application communication and additionally restricts the number of concurrent data transfers. In our past evaluations we found that the PA_Con_1 scheduler had the least impact on the application runtime.

B. Data Extraction Performance

To examine the impact on CHIMERA performance caused by background data movement, we compare the total execution time of the CHIMERA simulation with Datatap I/O and pHDF5. For Datatap, the visible I/O overhead is the total I/O blocking time in restart dumps plus the time of one-time finalization (during which all compute nodes block waiting for servers to fetch the data).

As shown in Figure 2, data extraction with the Datatap outperforms those with pHDF5. The time breakdown (shown in Figure 3) reveals that the improvement of total execution time is due to reduction of blocking I/O time and that the computation (main loop time) is not affected.

In terms of cost/effectiveness, Table I shows that at the scale of 8192 cores, the I/O overhead with pHDF5 is 37.84%. By using 16 additional compute nodes (64 cores in total) for staging, the I/O overhead is reduced to 0.14%. The additional Datatap servers only cost 64/8192=0.78% additional resources.

In the past we have presented data extraction performance with up to 2k processing cores [2] showing significant performance benefits compared to traditional POSIX output especially when using managed I/O.

TABLE I
VISIBLE I/O OVERHEAD

Overhead(%)	512	1024	2048	4096	8192
Datatap	0.0221	0.0248	0.0741	0.142	0.144
pHDF5	7.123	8.925	13.941	22.551	37.837

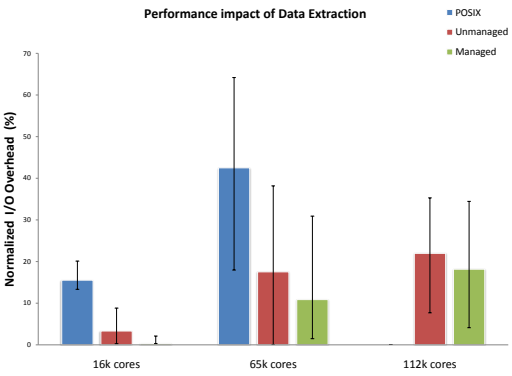


Fig. 4. Data Extraction overhead for GTC on the Cray XT5

In Figure 4 we look at the percentage performance overhead compared to a NULL data transport as we scale from 16,384 to 114,688 processing cores. As the number of processing cores increases from 16k to 65k, the performance overhead increases from 15.5% to 42.5%. Since Jaguarpf is a shared resource, we could not complete a run for POSIX at 112k cores. We plan to include the numbers for POSIX at 112k cores in the final version of the paper. For data extraction using the Datatap using a combination scheduling policy we found that the performance overhead was as low as below 1% at 16k cores to only 18.1% at 112k cores.

Figure 5 shows the distribution of blocking times for one representative I/O output over all the nodes in the system.

Although data extraction to the Datatap server provides a significant performance benefit compared to traditional POSIX I/O, the importance of providing a control infrastructure can be seen from the distribution of blocking time. The unmanaged data stream completes all transfers before the start of the next I/O phase, thus almost 100% of the nodes show a blocking time of less than 2 seconds. However the managed data transfer limits the time periods in which data can be moved out of the compute nodes. This results in a small number of nodes (about 10%) taking longer than 10 seconds blocking for the transfer to complete. A tiny fraction, about 1% block waiting for transfer completion for longer than 90 seconds. So despite the greatly reduced perturbation impact from the managed stream, the overall performance improvement is not as significant. This is an example of a scenario where an independent control plane that optimizes the data movement management for

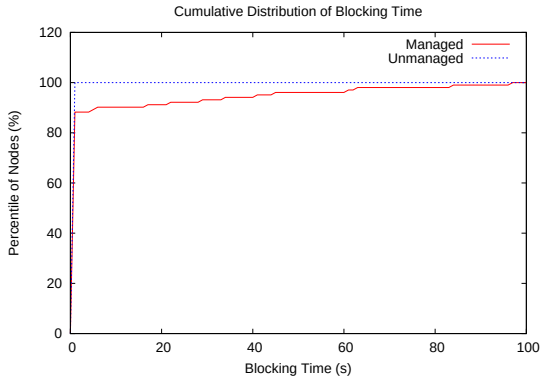


Fig. 5. Cumulative distribution of blocking time for a representative I/O phase with 112k processing cores

individual application runs can be used for large gains in performance.

C. Data Processing Performance

We evaluated the benefits of online data processing using data formatting is for CHIMERA as an example. The data formatting service produces HDF5 formatted data output by utilizing the computational resources of the staging area used for data extraction. Writing data into HDF5 requires first decoding FFS serialized data, combining local arrays into larger chunks, and then writing data to HDF5 file. Figure 6 shows the time breakdown of data formatting. Note that the FFS decoding time shown in Figure 6 is the total time of decoding 512 data chunks. Decoding one message (930KB) takes 0.091 seconds on average. The time of combining data chunks is about 0.25 seconds in total within one dump. These two parts are constant among all tests since the data size is kept unchanged.

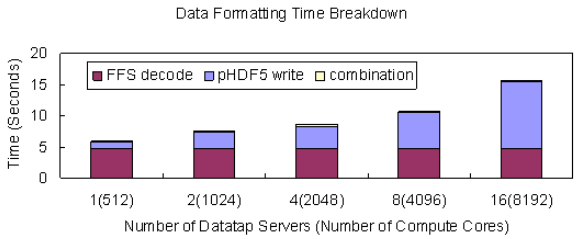


Fig. 6. Data Formatting Time

However the time spent in pHDF5 API increase linearly with the total size of the output data. As shown in Figure 7, the egress bandwidth increases only slightly as we increase the number of Datatap servers. This is due to the increase in nodes resulting in an equivalent increase in the coordination and synchronization costs associated with writing a HDF5 file as observed in our previous work [21]. The ingress bandwidth, however, increases more rapidly as we scale the overall application size. The less than linear increase is due to the increased network contention as more nodes are added into the cohort.

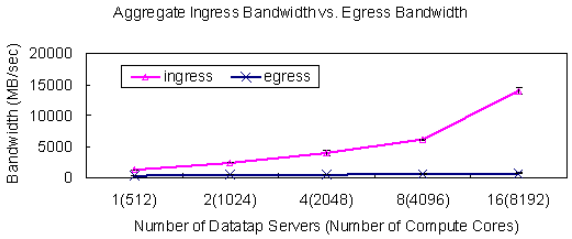


Fig. 7. Aggregate ingress and egress bandwidth. The egress bandwidth is limited by both the processing complexity of the HDF5 conversion and the utilizable filesystem bandwidth.

Another metric of interest is the total data processing time, which is the time period from getting the first request till finishing writing all data to HDF5 file. The total data processing time at different scales is shown in Figure 8. At the scale of 8192 compute cores, background processing on Datatap servers can be done in less than 20 seconds, while the restart interval is about 400 seconds. This suggests that there is sufficient time for background processing without blocking computation.

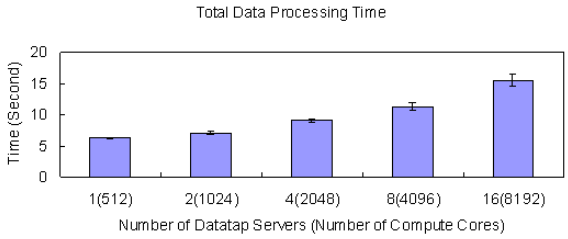


Fig. 8. Total Data Processing Time

D. Scheduled I/O Timing

To evaluate scheduled I/O, the system performance is evaluated using different processor counts, stripe counts, and stripe sizes using standard MPI-IO calls. To keep these measurements consistent, the stripe sizes are set so that no single process will cross a storage target boundary during a single write. The stripe count varies as indicated in the figure. The number of files generated is based on the total number of storage targets, 672 on this machine, divided by the stripe count. Each file is assigned a non-overlapping range of storage targets. To achieve less interference, the processes assigned to each storage target are serialized using MPI messages to signal the next process it is safe to write.

For the first set of tests, the write size is slightly larger than 128 MB. Figure 9 shows the average performance for different process counts and stripe counts for these

tests. The 'base' case is a default stripe count and stripe size to a single file. The error bars indicate the range of performance values measured. The key observations for this figure is that the maximum performance is obtained using a stripe count of 1, but the best average performance is obtained using a stripe count of 3.

In a second series of tests, the write size is increased to slightly more than 768 MB per write, simulating the notion of the SmartTap servers collecting the output from several cores and then writing in a single, larger chunk. As with before, the 'base' case is a default stripe count and stripe size to a single file. Unlike before, figure 10 shows the best average and overall write performance is obtained with a stripe count of 1. This difference further motivates the need for managing the I/O more than just using a staging area for optimal performance. By exploiting this knowledge, fewer staging resources can be employed to obtain the same or better overall system performance.

V. Conclusion and Future Work

Data Services have been presented as a useful abstraction for allowing end users to address the problems that come from I/O at extreme scale. By decoupling the management of the I/O transport and storage from the application interface, portability and robustness can be improved for most end users. Additionally, offering storage as a service opens up the possibility for a greater range of functionality within the I/O pipeline, including indexing, selection, or map-reduce-type manipulations.

In particular, this work has demonstrated that the separation of extraction, data manipulation, and management of the data fast path can offer significant performance improvements. Scale measurements over 100k processes show that, as expected, management issues change as the scale of the problem increases. This work demonstrates that using a combination of management techniques, including asynchronous transport, in-transit filtering of the data, and distributed synchronization, our implementations of data services can perform at or substantially better than the corresponding POSIX-based implementation.

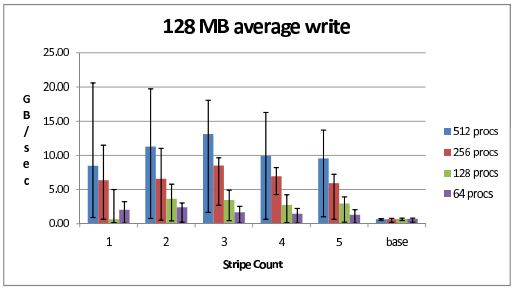


Fig. 9. 128 MB Writes

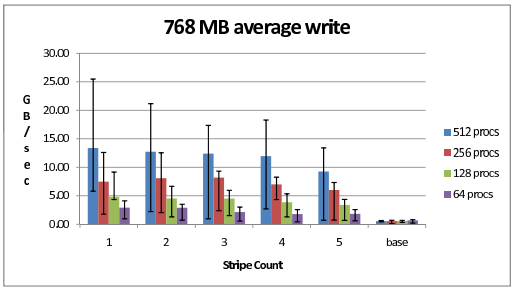


Fig. 10. 768 MB writes

This represents beginning work on the Data Service abstraction, and a number of interesting directions exist. Careful exploration of what should or should not be characterized as a data service (as opposed to a true post-processing workflow) so that total throughput guarantees can be maintained is one just direction. Runtime mechanisms to autonomically specialize and adapt the placement and routing of the I/O streams based on the changing resource availability are another. Finally, investigating the techniques for describing/programming the Data Services for scalable, portable deployment above 100k nodes will be a key space for innovation.

References

[1] H. Abbasi, M. Wolf, and K. Schwan. LIVE Data Workspace: A Flexible, Dynamic and Extensible Platform for Petascale Applications. *Cluster Computing, 2007. IEEE International*, Sept. 2007.

[2] Hasan Abbasi, Greg Eisenhauer, Matthew Wolf, and Karsten Schwan. A high performance infrastructure for composable event overlays. To appear in the Proceedings of HPDC 2009.

[3] Sandip Agarwala, Greg Eisenhauer, and Karsten Schwan. Morphable messaging: Efficient support for evolution in distributed applications. In *Challenges of Large Applications in Distributed Environments (CLADE)*, June 2004.

[4] Michael Beynon, Renato Ferreira, Tahsin M. Kurc, Alan Sussman, and Joel H. Saltz. Datacutter: Middleware for filtering very large scientific datasets on archival storage systems. In *IEEE Symposium on Mass Storage Systems*, pages 119–134, 2000.

[5] Fabian E. Bustamante, Greg Eisenhauer, Karsten Schwan, and Patrick Widener. Efficient wire formats for high performance computing. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 39. IEEE Computer Society, 2000.

[6] Z. Cai, V. Kumar, and K. Schwan. Iq-paths: Self-regulating data streams across network overlays. In *Proceedings of the IEEE Symposium on High Performance Distributed Computing, Paris, France*, 2006.

[7] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev Thakur. PVFS: A parallel file system for linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, 2000. USENIX Association.

[8] Lustre: A scalable, high-performance file system. Cluster File Systems Inc. white paper, version 1.0, November 2002. http://www.lustre.org/docs/whitepaper.pdf.

[9] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[10] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Kent Blackburn, Albert Lazzarini, Adam Arbree, Richard Cavanaugh, and Scott Koranda. Mapping abstract complex workflows onto grid environments. *J. Grid Comput.*, 1(1):25–39, 2003.

[11] C. Docan, M. Parashar, and S. Klasky. High speed asynchronous data transfers on the cray xt3. In *Cray User Group Conference*, 2007.

[12] Greg Eisenhauer. The evpath library. http://www.cc.gatech.edu/systems/projects/EVPath.

[13] Greg Eisenhauer. Portable binary input/output. http://www.cc.gatech.edu/systems/projects/PBIO.

[14] Greg Eisenhauer. The ECho Event Delivery System. Technical Report GIT-CC-99-08, College of Computing, Georgia Institute of Technology, 1999. *(http://www.cc.gatech.edu/tech_reports)*.

[15] Greg Eisenhauer, Fabian Bustamente, and Karsten Schwan. Event services for high performance computing. In *Proceedings of High Performance Distributed Computing (HPDC-2000)*, 2000.

[16] I. Foster, J. Vockler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on*, pages 37–46, 2002.

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

[18] Rob Latham, Neil Miller, Robert Ross, and Phil Carns. A next-generation parallel file system for linux clusters. *LinuxWorld*, 2(1), January 2004.

[19] Jay Lofstead, Fang Zheng, Scott Klasky, and Karsten Schwan. Input/output apis and data organization for high performance scientific computing. In *In Proceedings of Petascale Data Storage Workshop 2008 at Supercomputing 2008*, 2008.

[20] Jay Lofstead, Fang Zheng, Scott Klasky, and Karsten Schwan. Adaptable, metadata rich io methods for portable high performance io. In *In Proceedings of IPDPS'09, May 25-29, Rome, Italy*, 2009.

[21] Jay F. Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. Flexible io and integration for scientific codes through the adaptable io system (adios). In *CLADE '08: Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, pages 15–24, New York, NY, USA, 2008. ACM.

[22] Bertram Ludscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, 2006.

[23] O. E. B. Messer, S. W. Bruenn, J. M. Blondin, W. R. Hix, A. Mezzacappa, and C. J. Dirk. Petascale supernova simulation with CHIMERA. *Journal of Physics Conference Series*, 78(1):012049–+, July 2007.

[24] Ron A. Oldfield, Arthur B. Maccabe, Sarala Arunagiri, Todd Kordenbrock, Rolf Rie sen, Lee Ward, and Patrick Widener. Lightweight I/O for Scientific Applications. In *Proc. 2006 IEEE Conference on Cluster Computing*, Barcelona, Spain, September 2006.

[25] Ron A. Oldfield, Patrick Widener, Arthur B. Maccabe, Lee Ward, and Todd Kordenbrock. Efficient Data Movement for Lightweight I/O. In *Proc. 2006 Workshop on high-performance I/O techniques and deployment of Very-Large Scale I/ O Systems (HiPerI/O 2006)*, Barcelona, Spain, September 2006.

[26] Eric Riedel, Garth Gibson, and Christos Faloutsos. Active storage for large-scale data mining and multimedia. In *Proceedings of the* 24^{th} *International Conference on Very Large Databases*, August 1998.

[27] Erik Riedel, Christos Faloutsos, Garth A. Gibson, and David Nagle. Active disks for large-scale data processing. *IEEE Computer*, 34(6):68–74, June 2001.

[28] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, 2002.

[29] K. E. Seamons, Y. Chen, P. Jones, J. Jozwiak, and M. Winslett. Server-directed collective i/o in panda. In *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, page 57, New York, NY, USA, 1995. ACM.

[30] Balasubramanian Seshasayee and Karsten Schwan. Mobile service overlays: reconfigurable middleware for manets. In *MobiShare '06: Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking*, pages 30–35, New York, NY, USA, 2006. ACM.

[31] NTB Stone, D. Balog, B. Gill, B. Johan-SON, J. Marsteller, P. Nowoczynski, D. Porter, R. Reddy, JR Scott, D. Simmel, et al. PDIO: High-performance remote file I/O for Portals enabled compute nodes. *Proceedings of the 2006 Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, June*, 2006.

[32] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[33] WX Wang, Z. Lin, WM Tang, WW Lee, S. Ethier, JLV Lewandowski, G. Rewoldt, TS Hahm, and J. Manickam. Gyrokinetic simulation of global turbulent transport properties in tokamak experiments. *Physics of Plasmas*, 13:092505, 2006.

[34] Patrick M. Widener, Matthew Wolf, Hasan Abbasi, Matthew Barrick, Jay Lofstead, Jack Pullikottil, Greg Eisenhauer, Ada Gavrilovska, Scott Klasky, Ron Oldfield, Patrick G. Bridges, Arthur B. Maccabe, and Karsten Schwan. Structured streams: Data services for petascale science environments. Technical Report TR-CS-2007-17, University of New Mexico, Albuquerque, NM, November 2007.

[35] Matthew Wolf, Hasan Abbasi, Benjamin Collins, David Spain, and Karsten Schwan. Service augmentation for high end interactive data services. In *IEEE International Conference on Cluster Computing (Cluster 2005)*, September 2005.

[36] Jia Yu and Rajkumar Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.*, 34(3):44–49, 2005.