PreDatA – Preparatory Data Analytics on Peta-Scale Machines

Fang Zheng*, Hasan Abbasi*, Ciprian Docan†, Jay Lofstead*, Qing Liu‡, Scott Klasky‡
Manish Parashar†, Norbert Podhorszki‡, Karsten Schwan* and Matthew Wolf*‡
*College of Computing, Georgia Institute of Technology, Atlanta, GA 30332
†Center for Autonomic Computing, Rutgers University, Piscataway, NJ 08854
‡Oak Ridge National Laboratory, Oak Ridge, TN 37831

***Abstract*—Peta-scale scientific applications running on High End Computing (HEC) platforms can generate large volumes of data. For high performance storage and in order to be useful to science end users, such data must be organized in its layout, indexed, sorted, and otherwise manipulated for subsequent data presentation, visualization, and detailed analysis. In addition, scientists desire to gain insights into selected data characteristics 'hidden' or 'latent' in these massive datasets while data is being produced by simulations. PreDatA, short for Preparatory Data Analytics, is an approach to preparing and characterizing data while it is being produced by the large scale simulations running on peta-scale machines. By dedicating additional compute nodes on the machine as 'staging' nodes and by staging simulations' output data through these nodes, PreDatA can exploit their computational power to perform select data manipulations with lower latency than attainable by first moving data into file systems and storage. Such in-transit manipulations are supported by the PreDatA middleware through asynchronous data movement to reduce write latency, application-specific operations on streaming data that are able to discover latent data characteristics, and appropriate data reorganization and metadata annotation to speed up subsequent data access. PreDatA enhances the scalability and flexibility of the current I/O stack on HEC platforms and is useful for data pre-processing, runtime data analysis and inspection, as well as for data exchange between concurrently running simulations.**

I. Introduction

Scientific applications running on High End Computing (HEC) platforms can generate large volumes of output. As these grow to peta-scale and beyond, fast write and read accesses to massive data are becoming increasingly important, both to speed up the simulation and to accelerate exploration of data. A prerequisite to data exploration is that data is prepared in terms of data layout, indexing, and annotation. For example, some analysis tools prefer data to be laid out as contiguous arrays for quick loading [49], and queries can be accelerated if data is properly sorted and indexed [42]. In other words, appropriate data preparation is critical for data analytics, inspection, or visualization to operate efficiently. Finally, 'hidden' in the voluminous data sets generated by running simulations are latent data characteristics of interest to end users, an example being statistical measures that can be used to validate the veracity of the ongoing simulation, gain understanding of the simulation progress, and potentially, take early action when the simulation operates improperly [20].

The object of our research and topic of this paper is the development of efficient methods that properly prepare data for subsequent inspection, storage, analytics, and even for input into concurrent, coupled simulation models (e.g., as in climate modeling). Our approach associates such data preparation with the output actions taken by simulations in ways that speed up output actions, thereby also improving simulation performance. The software artifact developed and used for these purposes is the PreDatA middleware. PreDatA provides scalable and flexible ways of associating data preparation operations with the I/O actions of HEC applications, by generalizing the I/O stack used by HEC codes and taking advantage of the ADIOS I/O library [29] used in a wide variety of peta-scale codes. The enhanced I/O stack enables efficient operations on output data via predefined or user-provided computational functions. These functions are performed while I/O is ongoing by staging data to where PreDatA can leverage the computational power of selected machine nodes supporting I/O and/or connected to the storage subsystems. Further, by using PreDatA to index or properly annotate data, a reduction in the volume of subsequent reads performed by scientific workflows engaged in data analysis can be achieved. This also reduces interference at the parallel file system due to simultaneous writes used by output and reads used by scientific workflows.

The PreDatA middleware exploits the additional computational and memory resources provided by a staging area resident on the peta-scale machine. Output data are moved from compute to staging area nodes asynchronously to reduce write latency. PreDatA operations are applied to data prior to leaving the compute node and/or on data buffered in the staging area. The middleware provides a pluggable framework for executing user-defined operations such as data re-organization, real-time data characterization, filtering and reduction, and select analysis (or pre-analysis). These operations are specified in ways natural to the 'streaming' context in which they are used. Despite this rich functionality, PreDatA offers levels of performance not provided by current file system-based approaches to analyzing output data, as shown with extensive experiments in this paper.

PreDatA performance is evaluated with several production peta-scale applications on Oak Ridge National Laboratory's

Leadership Computing Facility platform. For one application, GTC [23], at the scale of 16,384 compute cores and with 1.5% additional resource usage, PreDatA hides write latency by up to 99.9%, improves total simulation time by 2.7%, and achieves a 1.5% saving in total CPU usage compared with performing pre-analytics in the compute nodes. In this experiment, PreDatA generates scientifically meaningful statistics from the 260GB output data in one simulation time step in about 40 seconds. For another application, Pixie3D [10], using PreDatA to re-organize the array layout of output data from 16,384 cores improves subsequent read performance for these output files by 10 times compared to when no such reorganization is performed. At the same time, total execution time of the simulation is improved by 1% with only 0.7% additional resource usage.

The remainder of the paper is organized as follows. Section II introduces the data management challenges for two motivating applications. Sections III and IV present the design and implementation of PreDatA, respectively. Section V applies the PreDatA approach to the two applications, and evaluates the resulting performance demonstrating its advantage over other online and/or offline approaches. Section VI summarizes related work, and Section VII concludes the paper.

II. Application Drivers

The development of PreDatA has been driven by the output and analysis needs of two production peta-scale codes, GTC and Pixie3D, both of which are capable of scaling to tens of thousands of cores and generating Terabytes of data in typical production runs.

A. The GTC Fusion Modeling Code

The Gyrokinetic Toroidal Code (GTC) [23] is a 3-Dimensional Particle-In-Cell code used to study microturbulence in magnetic confinement fusion from first principles plasma theory. It outputs particle data that includes two 2D arrays for electrons and ions, respectively. Each row of the 2D array records eight attributes of a particle including coordinates, velocities, weight, and particle label. The last two attributes, process rank and particle local ID within the process, jointly form the label that globally identifies a particle. They are determined on each particle at the start of a simulation and remain unchanged throughout the particle's lifetime. These two arrays are distributed among all cores, and particles move across cores in a random manner as the simulation evolves, resulting in two out-of-order particle arrays. In a production run at the scale of 16,384 cores, each core can output two million particles roughly every 120 seconds resulting in 260GB of particle data per output.

As shown in Fig. 1, three analysis and preparation tasks are performed on particle data. The first involves tracking across multiple iterations of a million-particle subset out of the billions of particles, requiring searching among the

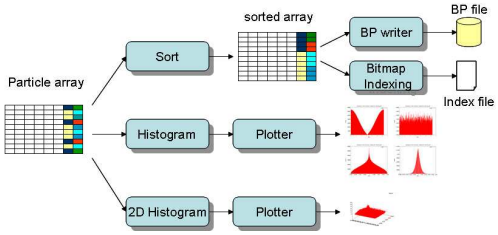


Figure 1. Illustration of PreDatA Operations on GTC Particle Data

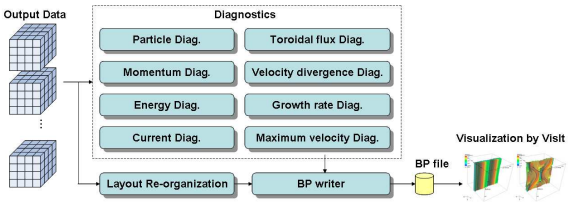


Figure 2. Illustration of PreDatA Operations on Pixie3D Output Data

hundreds of 260GB output files by the particle label. To expedite this operation, particles can be (and for our example are) sorted by their labels before searching. The second task performs a range query to discover the particles whose coordinates fall into certain ranges. A bitmap indexing technique [42] is used to avoid scanning the whole particle array, and multiple array chunks are merged to speed up bulk loading. The third task is to generate 1D histograms and 2D histograms on attributes of particles [21] to enable online monitoring of the running GTC simulation. 2D histograms can also be used for visualizing parallel coordinates [21] in subsequent data analysis.

B. The Pixie3D Code

Pixie3D [10] is a 3-Dimensional MHD (Magneto Hydro-Dynamics) code that solves the extended MHD equations in 3D arbitrary geometries using fully implicit Newton-Krylov algorithms. Pixie3D employs multigrid methods in computation and adopts a 3D domain decomposition. The output data consists of eight 3D arrays that represent mass density, linear momentum components, vector potential components, and temperature, respectively.

As illustrated in Fig. 2, various diagnostic routines are performed on Pixie3D output data to generate derived quantities such as energy, flux, divergence, and maximum velocity. These derived quantities, along with the raw output data, are then read by visualization tools like VisIt for interactive visual data exploration. Pixie3D employs the BP file format for fast write performance [29]. Array layout re-organization is performed to speed up subsequent read access.

C. Using the Staging Area for Flexible Scalable I/O and Pre-Data Analytics

Conventionally, data preparation and analytics are performed either in compute nodes where the simulation is running or offline:

In-Compute-Node approach: operations are performed in the compute nodes where output data is generated. The processed output is then written to the parallel file system.

Offline approach: the simulation dumps data to a parallel file system. Analysis codes running on other resources read such data and operate on it.

These two approaches to processing simulation output data differ in terms of their respective costs and limitations. For the In-Compute-Node approach, the overhead of data processing operations is visible to the simulation. This has consequent expenses in terms of CPU hours at scale and may require additional application tuning. Performance advantages result if In-Compute-Node actions reduce output volumes, but severe performance penalties arise if data processing operations do not scale with the simulation. For the Offline approach, if the data volume is large, intermediate files may consume considerable storage resources, and parallel file system write and read times can be dominant causing high latencies and unacceptable levels of perturbation of peak file system performance. These arguments motivate the need for additional methods to satisfy the I/O and data processing needs of these two representative peta-scale codes.

The *Staging Area* approach to satisfying peta-scale I/O needs uses a reasonable number of compute nodes as a 'Staging Area' for staging data and to host operations that are applied to staged data before it reaches storage. Asynchronous execution within the Staging Area hides its processing costs from the simulation, and it also permits users to use less scalable analysis codes 'at scale', the latter due to the fact that the Staging Area is small in comparison to the number of compute nodes used for simulation (e.g., using a ratio of 128:1 for compute vs. stagings cores). Staging Area codes can also reduce disk accesses by pre-processing data so that later analytics can focus on the data that is most relevant. Finally, since the staging nodes are tightly coupled with compute nodes, data movements to staging can be done efficiently [2]. The PreDatA middleware presented in this paper exploits these facts by running in a Staging Area on peta-scale machines.

III. PreDatA Middleware Design

The PreDatA middleware design augments the current I/O stack on HEC platforms with data staging and in-transit processing capabilities. As shown in Fig. 3, PreDatA middleware resides on both staging nodes and the compute nodes on which the application runs. When the application performs I/O actions, PreDatA acquires output data through the ADIOS I/O interface [28], stages data from compute nodes to staging nodes and performs in-transit data processing along the data flow. Its current implementation exploits the computational resources of compute nodes for data selection and movement and those of the Staging Area for preparatory data analytics.

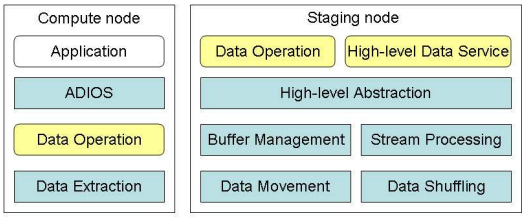


Figure 3. PreDatA Middleware Architecture

There are several key features of PreDatA:

Asynchronous data movement. Data movement from compute to staging nodes is performed asynchronously to hide write latency from the simulation, at moderate consequent costs for data buffering on compute nodes. PreDatA explicitly schedules asynchronous data movement to minimize interference with the simulation's communications.

Pluggable pre-data analytics. PreDatA provides a pluggable framework, making it straightforward for end users to specify, deploy, and debug data processing operations. The programming interface is general enough to implement a variety of operations, including data re-organization, real-time data characterization, filtering and reduction, and lightweight data analysis.

User-defined operations. The middleware supports user-defined data operations by providing basic services for data access, buffer management, scheduling and executing data processing actions, and for high performance data exchange and synchronization across staging nodes.

Higher-level services. The middleware also offers select higher-level data services, such as those for data indexing and for the data queries needed for inter-application data exchanges.

IV. PreDatA Middleware Implementation

The PreDatA middleware's implementation leverages our earlier work [2] on efficiently scheduling data movement from compute nodes to the Staging Area. The EVPath [17] high performance event system is used for efficient data buffering and manipulation in the Staging Area. The FFS [18] binary data encoding facility is used for in-transit data to provide PreDatA operations access to buffered data with rich meta-data information.

A. Data Extraction and Movement

PreDatA uses the ADIOS I/O library [28] for integration with the HEC I/O stack and in addition, for PreDatA operations to access the data output by simulations. With ADIOS, PreDatA processing can be added without requiring changes to application codes, thereby insulating application code from the complexities of additional processing actions in the I/O stack. ADIOS also explicitly defines the structure of the application's output data, and such meta-data is used as a common interface for application and PreDatA operations to coordinate sharing data.

Data is extracted from compute nodes and moved to the Staging Area via the scheduled, asynchronous RDMA [7] operations. As explained in [2], using asynchronous RDMA reduces the write latency visible at compute nodes. Carefully scheduling such RDMA operations eliminates the potential interference between communications performed by the simulation vs. those used for output. This is particularly important when output data movement overlaps collective communications among compute nodes and thereby may cause severe perturbation on simulation performance.

B. In-transit Data Processing along the Data Flow

PreDatA augmentation of the I/O stack results in the overall data flow shown in Fig. 4. There are four stages in the data flow: (1) data extraction and optional local processing in compute nodes, (2) optional aggregation in staging nodes, (3) asynchronous data movement from compute nodes to staging nodes, and (4) data stream processing in staging nodes.

When I/O is triggered in the compute nodes, output data is passed to the PreDatA runtime in the compute nodes (shown as Stage 1 in Fig. 4). Typical output data consists of one or more scalars, local arrays, and/or partial chunks of global arrays. PreDatA executes a user-defined routine *Partial_calculate()*, if provided, on the local output data (shown as Stage 1a in Fig. 4). This constitutes an optional first pass of processing on the output using local and typically deterministic (in terms of delay) operations. Examples include generating meta-data such as array dimension information, calculating local min/max values of partial array chunks, and filtering out undesired regions. All output data (scalars, local arrays, partial chunks of global arrays) are then packed into a contiguous buffer, termed a *packed partial data chunk*, using the FFS [18] binary data encoding facility (shown as Stage 1b in Fig. 4). The structure of each packed partial data chunk is compatible with the ADIOS output data group definition, and metadata about the data structure is embedded in the packed partial data chunk. A data fetch request is sent to the staging node chosen by a user-overridable function *Route()* (shown as Stage 1c in Fig. 4). PreDatA provides an interface that permits the data operation in Stage 1a to attach small partial results to data fetch requests, allowing for additional flexibility in the staging area. The compute node then resumes computation while the data movement and operations are performed.

In the Staging Area, each node waits for data fetch requests from compute nodes. When the staging node finishes gathering requests from all of the compute nodes it serves, it extracts partial results attached to requests, if any are present, and it then applies user-defined aggregation functions to them to generate aggregated results, such as global array sizes and offsets, prefix sums, and global min/max values (shown as Stage 2 in Fig. 4). Next, each staging node begins to fetch packed partial data chunks from compute nodes

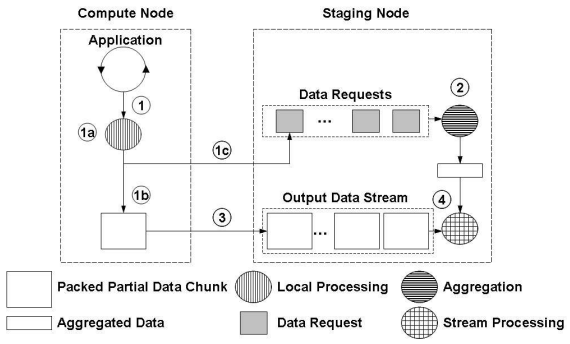


Figure 4. Overall Data Flow of PreDatA

(shown as Stage 3 in Fig. 4). Data chunks are processed by staging nodes one by one in a streaming manner (shown as Stage 4 in Fig. 4). The aggregated results generated in Stage 2 are accessible to such stream processing operations.

In summary, PreDatA provides two passes across an application's output data. The first pass optionally done on compute nodes is suitable for operations which have deterministic delays and do not require global communications and/or synchronization. The second pass performed on staging nodes, in a data streaming fashion, can be used to compute global data properties and/or to reorganize data for later storage. Data streaming is critical because it is unlikely for staging nodes to have sufficient memory to hold all of the raw data generated by multiple and, often, even single simulation output steps.

C. Stream Processing in the Staging Area

As mentioned above, the output data of each compute node is packed into a contiguous memory buffer, i.e., a *packed partial data chunk* and moved in its entirety into the Staging Area. From the Staging Area's perspective, incoming data consists of a finite number of packed partial data chunks streamed from compute nodes participating in the I/O dump. When there are multiple staging nodes, the packed partial data chunks are split into multiple streams across these nodes.

Each staging node is responsible for processing a stream of packed partial data chunks, with each chunk from one compute process. This is the forth stage of the dataflow shown in Fig. 4. The processing of such a stream is divided into five phases (as shown in Fig. 5):

Initialize: the *Initialize()* function of each operation is executed once at the beginning of an I/O dump, with aggregated result data generated from the pre-fetch process (shown as Stage 2 in Fig. 4) as a parameter to initialize operation-specific data structures and for other setup tasks.

Map: the *Map()* function of each operation is executed on each packed partial data chunk. Intermediate results are tagged and stored in a local buffer.

Shuffle: when the last chunk within the I/O dump is processed, partial results are combined locally, if the *Combine()* function is provided. Each staging node applies the

Partition() function to route intermediate results to other staging nodes according to the associated tags.

Reduce: each staging node groups intermediate results, both local and those received from other staging nodes, by associated tags and then performs the *Reduce()* function on each group of intermediate results to aggregate results.

Finalize: when the Reduce phase finishes, each staging node executes the *Finalize()* function of each operation, which writes final results to disk, feeds data to other consumers, and/or performs necessary cleanup.

From this description, it should be apparent that the PreDatA data processing model is similar to the MapReduce [12] paradigm, with four notable differences. First, PreDatA's data processing model requires operations to read data only once, meaning that data is processed in a streaming fashion, as done in other streaming implementations of MapReduce [11]. This is because of limited memory space on staging nodes, which means that this assumption can be removed if additional memory (e.g., SSD or other disk storage) were to be made available on those machines. Second, PreDatA adds Initialize and Finalize phases in order to deal with input from the application and output to storage or data transfer to remote nodes, respectively. Third, in contrast to the MapReduce model, analysis operations running in the staging area can implement customized data shuffling and synchronization methods, in our case using the highly-optimized MPI routines present on the peta-scale machine (see [53]). This is not only to take advantage of the high end communication support available on the peta-scale machine but also to be able to deploy and leverage existing parallel analysis codes written for science applications. Fourth, the PreDatA implementation does not use a central master with global knowledge of data location and task progress, exploiting the extensive prior knowledge and experience in the HPC community with how to program and implement efficient parallel codes.

Custom data operations are plugged into PreDatA middleware by implementing the functions mentioned above. Users can also customize data movement scheduling policies and to place the data chunks present within the data stream into some desired order to ease implementing such data analysis services. More details about the programming interface can be found in [53].

The staging area is running as a separate MPI program launched independently from the simulation. Each MPI process runs on one staging node. Within each staging node, there are multiple threads in each MPI process that exploit concurrency in how they execute different parts of the execution flow shown in Fig. 5.

D. The DataSpaces Global Data Knowledge Service

The purpose of this section is to show that PreDatA's 'in-transit' and 'online' approach to data output and manipulation can be used to implement the model-to-model communications used in coupled high performance codes [3], [51]. Toward that end, we have integrated into PreDatA the 'DataSpaces' data indexing and querying services. DataSpaces provides high level programmable and managed services for (1) data sharing – between operations working on a common set of data; (2) data redistribution – between operations with different data discretization and running on a different number of processors; (3) data indexing – data hashing for fast access; and (4) data querying – application data retrieval based on custom selectors. With (1)–(4), it provides the abstraction of a virtual semantically-specialized shared data space that can be asynchronously and flexibly accessed using simple yet powerful operators (e.g., *put()* and *get()*) that are agnostic of the location or distribution of data.

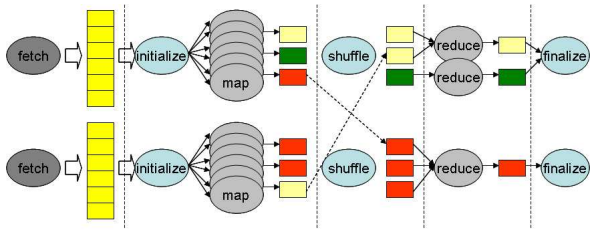


Figure 5. Stream Processing in the Staging Area

DataSpaces incorporates flexible mechanisms that can fetch and index data, on-the-fly, from multiple different sources, as shown in Fig. 6. It can store incoming data locally in the Staging Area or share it with the collaborating frameworks, index it for fast access, and serve it in response to logged or incoming user queries. Datasets composed of both, homogeneous data types, e.g., doubles, floats or integers, as well as heterogeneous data types, e.g., aggregate structures of doubles, floats or integers, are supported.

DataSpaces implements a flexible querying mechanism that allows applications to request individual values as well as contiguous regions of data based on simple descriptors that are semantically meaningful to the application. For example, simulation data can be indexed based on its geometric coordinates within the multi-dimensional discretization used by the simulation, allowing it to be queried using geometric descriptors that are meaningful to the application. Queries may be generated by users or by other applications. For example, each instance of a distributed querying application running on multiple nodes can query distinct and relevant sub-regions of data as needed. Similarly, a user can query sub-regions of interest only when they are needed or can register sub-regions of interest for continuous querying. In the latter case, the user is notified automatically every time new data items that lie within the regions of interest are inserted into the space.

DataSpaces also supports aggregation and reduction queries, e.g., the max/min/average value for a particular field in a given sub-region. From the perspective of a querying end user or application, the querying and data transfer process is transparent and independent of the data

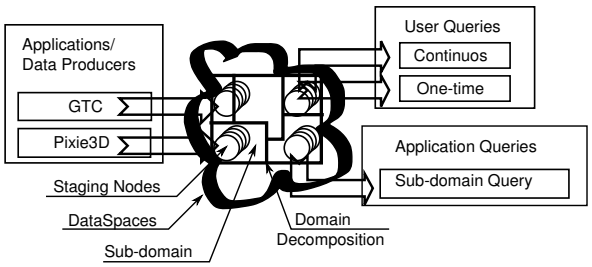


Figure 6. Example Scenario of Model to Model Coupling with DataSpaces

distribution, i.e., the data comprising the query response may come from different nodes of the application that generated the data and served by different DataSpaces framework nodes.

DataSpaces complements the indexing and querying services with an in-memory data storage service. The storage service can maintain private copies of the data extracted directly from a running application or store shared copies of the data processed by collaborating frameworks. The storage service incorporates a data coherency protocol that manages interactions with the data and ensures data integrity when multiple entities simultaneously query the data.

DataSpaces maintains load balancing at two levels. First, the storage service distributes the data evenly across the DataSpaces nodes, and second, the indexing service dynamically distributes the index metadata across the DataSpaces nodes to distribute incoming queries across these nodes.

V. Performance Evaluation

The placement of PreDatA operations can greatly affect the performance of PreDatA analyses as well as the timeliness of simulation output, thereby potentially greatly impacting overall system performance. The experimental evaluations shown in this section evaluate different PreDatA operators and different placement choices. The goal is to demonstrate the potential benefits from using the PreDatA approach and to show that whenever data is processed in-transit, it is important to be flexible in where the operators performing such processing are placed. Experimentation uses two driver applications described in Section II. Sorting, histogram, and 2D histogram operators are tested for GTC, where processed particle data is then written into storage from the Staging Area. For Pixie3D, an array layout reorganization operation is created. This operation merges partial array chunks into larger contiguous ones for each of the eight 3-dimensional arrays in Pixie3D's output, and it then writes merged arrays to output. The performance of the DataSpaces global data knowledge service is also evaluated with GTC. This demonstrates the feasibility of building higher-level data services with PreDatA. For brevity, the implementation details about those operations appear in [53].

A. Experimental Environment

Experiments are run on the Oak Ridge National Laboratory's Cray XT4/XT5 Jaguar platform. The XT5 partition contains 18,688 compute nodes. Each compute node contains two quad-core AMD Opteron 2356 (Barcelona) processors running at 2.3 GHz, with 16GB of DDR2-800 memory, and an attached SeaStar 2+ router. The resulting partition contains 149,504 processing cores, more than 300TB of memory, over 6 PB of disk space, and has a peak performance of 1.38 petaflop/s. The XT4 partition contains 7,832 compute nodes. Each compute node contains a quad-core AMD Opteron 1354 (Budapest) processor running at 2.1 GHz, 8 GB of DDR2-800 memory,and a SeaStar2 router. The resulting partition contains 31,328 processing cores, has more than 62 TB of memory and over 600 TB of disk space, and offer peak performance of 263 teraflop/s. Each test case described below is run 5 times, in order to eliminate disturbances in performance measurements caused by the shared nature of our experimental environment (i.e., other HPC codes running on the same machine as well as read-based loads imposed on the shared file system). For this reason, the experimental results shown constitute the best samples in both the In-Compute-Node and the Staging configurations.

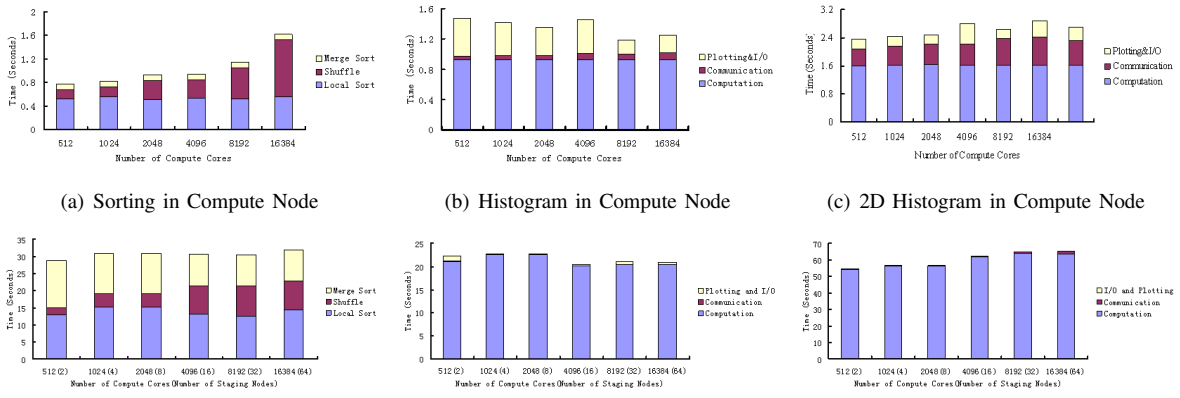
B. GTC Performance

The GTC experiments are performed on the XT5 partition of Jaguar. As is typical with a production run, the GTC jobs are configured to deploy a single MPI process per node that spawns 8 OpenMP worker threads, one per core. I/O is only performed by the MPI processes. For GTC, three operations are tested: particle sorting, histogram generation, and 2D histogram generation. Each of these operators is applied to both the electron and ion particle arrays that are output with an I/O interval of roughly 120 seconds. Weak scaling is employed, with 132MB total data written per process for the two particle arrays. The Staging Area is configured to deploy 2 MPI processes per node, with 4 worker threads per MPI process. The size of the Staging Area is adjusted to maintain a ratio of compute cores to staging cores of 64:1 (1.5%). That is, for each 64 nodes with compute processes (512 OpenMP worker threads), 1 node (2 staging processes for a total of 8 worker threads) is employed for staging.

Tests are performed in two ways. First, all operations are performed in compute nodes, using synchronous MPI-I/O to write results ('In-Compute-Node' configuration). Second, they are performed in the Staging Area ('Staging' configuration).

1) Performance of Individual Operations: In this section we study the performance results for each operation.

Sorting Operation: Fig. 7(a) and 7(d) compare the performance of sorting using the In-Compute-Node configuration and the Staging configuration. Sorting is communication-intensive because it involves all-to-all communication and



(a) Sorting in Compute Node (b) Histogram in Compute Node (c) 2D Histogram in Compute Node

(d) Sorting in Staging Area (e) Histogram in Staging Area (f) 2D Histogram in Staging Area

Figure 7. Timing Results for Individual Operations

has minimal computational demands. When sorting in compute nodes, the data shuffle time among compute nodes increases dramatically as the operation scales and such costs are visible to the simulation. In contrast, sorting in the Staging Area takes at most 33 seconds at all scales, which is much less than the 120-second I/O interval. This means that the asynchronous use of the Staging Area for sorting can mask sorting overheads from the simulation. There are, however, 30 seconds of latency in the Staging configuration, which is two orders of magnitude larger than the latency experienced in the In-Compute-Node configuration. This tradeoff demonstrates the importance of placement: if the goal is to optimize simulation time, placing the sorting operation into the Staging Area is better, but if the latency of generating sorted data is more critical, it is preferable to place the operator into compute nodes.

Histogram Operation: As shown in Figs. 7(b) and 7(e), the histogram operation is computation-dominant, with communication contributing only a very small portion of total operation time. While performing this style of computation-intensive operation in the compute nodes takes less wall clock time, perturbations to total simulation time can be much larger due to the impact of I/O operation for saving histogram results. The time for writing the 8 MB histogram files ranges from 0.25 seconds to 7 seconds, which adds to the total simulation time. This reveals a different advantage for the Staging configuration: its ability to insulate the simulation from variations in file system performance. Since the increased cost of generating the histogram is hidden by the asynchronous data transfer and operation, using the Staging configuration is advantageous. For those cases where one has computation-intensive operations without a subsequent I/O operation or if latency is very important, using the 'In-Compute-Node' configuration is superior.

2D Histogram Operation: As with the Histogram operation, the 2D Histogram is also computation-dominant, as shown in Figs. 7(c) and 7(f), implying that the conclusions drawn from this experiments are much like those of the previous one, modulo the fact that the computation and communication requirements for generating the 2D histograms are higher.

In summary, the results shown in this section demonstrate that it is often advantageous to offload PreDatA operations from compute to staging nodes, since such offloading can help mask from the simulation the costs and variations of such operations as well as those of the I/O activities associated with them. A potential detriment is the increase in operation latency because of the capacity mismatch between compute and staging nodes. Future work is needed to automate placement decisions, where automation would be based on higher level inputs from application developers and users and on information about current platform and file system states.

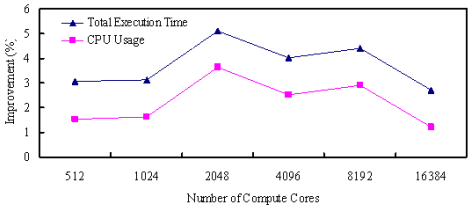
2) Simulation Performance: This section evaluates GTC simulation performance in two different configurations. Fig. 8(b) shows the total execution time of the GTC simulation for the two different configurations at various scales ranging from 512 to 16,384 compute cores. The Staging configuration improves the simulation's total execution time by 2.7% to 5.1% over the In-Compute-Node configuration (as shown in Fig. 8(a)).

The breakdown of total execution time (shown in Fig. 8(b)) explains the performance advantage of the Staging vs. In-Compute-Node approach. First, the Staging approach hides write latency via asynchronous data movement. For example, at the scale of 16,384 compute cores, 8.6 seconds are required, on average, to write 260GB of particle data with the ADIOS synchronous MPI-I/O method. The visible I/O blocking time with the Staging configuration is reduced to 0.30 seconds on average. This improvement in write latency increases with simulation scale. Second, the Staging approach also insulates the simulation from the increasing time costs for performing the operations as the simulation scales, since staging area code runs concurrently and asynchronously with the simulation. For the In-Compute-Node configuration, the time spent in operations increases from

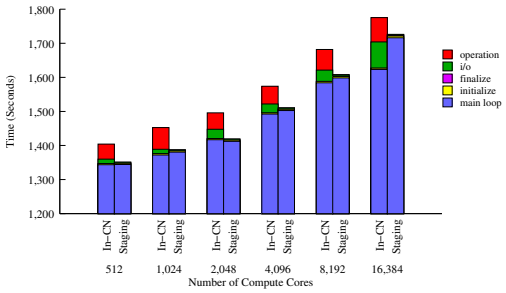
3.0% to 4.1% as the simulation scales from 512 to 16,384 cores. With the Staging approach, the simulation spends no time carrying out such operations. While it is true that the Staging Area experiences a larger proportional time in performing the operations, the additional latency easily fits into the simulation's current I/O period, thereby not affecting the application's execution time. Third, potential interference between asynchronous data movement with the simulation's communications can be minimized by properly scheduling data movement. Specifically, a comparison of main loop time for the two different configurations shows that staging may slow down the computation due to contention on the shared network, especially at large scales, but by properly scheduling data movement, this interference is controlled to be less than 6% in the worst case.

Overall, the reduction in visible I/O and operation times on compute nodes outweighs the interference experienced by the simulation due to asynchronous I/O, and the insulating effects of decoupling simulation I/O from variations of file system performance both improve total simulation time and reduce variations in simulation performance. These facts hold despite increased latencies for performing certain PreDatA operations. In terms of total CPU usage, calculated as total simulation time multiplied by total cores used, the Staging configuration is less costly compared to the In-Compute-Node configuration at all scales (as shown in Fig. 8(a)). There is some decline in these savings when scaling from 8,192 to 16,384 cores, mainly due to the interference of asynchronous data movement with the simulation's use of collectives. Despite this fact, at the scale of 16,384 cores, running the simulation with the Staging configuration still saves 98 CPU hours in total compared with the In-Compute-Node configuration, for a 30 minute simulation run. This suggests that the Staging approach helps GTC achieve better scalability in terms of total cost of both simulation and data preparation.

3) Offline Operation: End users may consider replacing PreDatA with offline operations, i.e., with operations applied to data after it has been written to disk. The following tradeoffs should be considered. First, doing so replaces the consumption of certain levels of compute and memory resources on the peta-scale machine with disk storage and file system use, the latter raising potential concerns for simulation performance due to file system interference by other jobs. Second, typically, offline operations, while slower to perform and exhibiting much higher latency to completion, can be done cheaply or 'free' (in terms of costs charged to end user accounts), the latter being an odd artifact of how peta-scale machine costs are imposed on HPC users. However, for operations that do not generate a reduction in data and instead, generate approximately equivalent data in a different organization, such as sorting and layout reorganization, an offline approach requires additional disk resources for intermediate data storage. In addition, it impacts



(a) GTC Simulation Performance and Cost Improvement



(b) GTC Total Execution Time Breakdown

Figure 8. GTC Simulation Performance

the file system due to repeated read/write of the data in question. For example, when running at the scale of 65,536 cores, the particle data of GTC is 1TB per I/O dump. Offline sorting would cost 1 TB additional storage space every 120 seconds, and the entire 1 TB would have to be read back to memory before it is rewritten. This moves the data through the disk controllers three times rather than once. Third, given the huge volume of GTC data, the read and write latency would be hundreds of seconds, making the offline approach unsuitable for the online monitoring functionality desired by GTC users.

In-transit, online data analysis is also preferable for operations like the histogram and 2D histogram. With the offline approach, using the same 1 TB per I/O dump output, two trips through the disk controller are required. While the output of these operations is comparatively small, reading all of the data to generate the histograms may cause both potentially large latency and long-term adverse impacts on file system performance.

4) Evaluation of the DataSpaces Global Data Knowledge Service: This section shows that the DataSpaces query engine can service queries on particle data in a timely manner, without blocking the simulation between two successive I/O operations. Experimental evaluations are performed with a prototype of the DataSpaces indexing and querying service deployed on the staging nodes. The particles output by the GTC application are first sorted using the sorting operation, and then indexed by DataSpaces, based on their *local id* and *rank* attributes, thereby creating a $2 \cdot 10^6 \times 256$ 2-D domain space. This space is then uniformly distributed across the DataSpaces compute cores in the Staging Area. On average, at all simulation scales ranging from 512 to 16,384 cores, the time required to fetch data from the GTC simulation is 20.3 seconds, sorting takes 30.6 seconds, and indexing takes 2.08

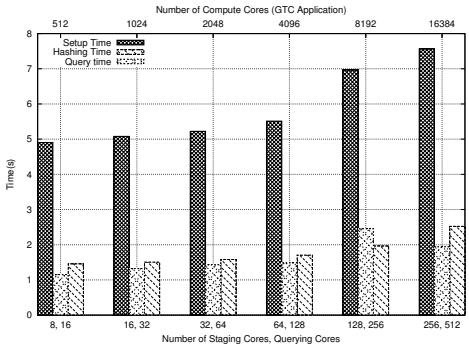


Figure 9. Setup, Hashing and Query Time

seconds. This means that in total, it takes no more than 55 seconds for DataSpaces to prepare the data for query, again well within the 120 second output period used by GTC.

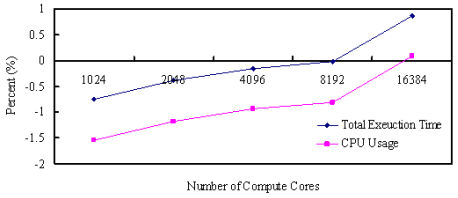
A test querying application that queries the entire domain space is deployed on additional compute cores (referred to as 'querying application cores' in the subsequent text). In our experiments, the querying application cores partition the particle data among themselves and issue 11 consecutive queries to disjoint regions of the data. The particle sub-regions are 200MB in size for each querying application core. Since no a-priori knowledge is assumed about the existence of the particle data or its distribution, the first query includes query setup operations, such as hashing, data discovery, query routing, and data retrieval. This means that it is significantly more expensive to perform, as seen in Fig. 9. However, this is a one-time cost and subsequent queries are much faster. The setup time shown in Fig. 9 is an average value across the number of querying application cores, and the hashing time is an average over the number of setup queries received at each core running a DataSpaces server in the Staging Area.

The query execution time for different numbers of querying application cores is also plotted in Fig. 9. The plotted times are an average over the number of queries executed and over the querying application cores. The query time increases with the number of cores used since the domain size increases and is mapped to a larger number of cores in the staging area. In our example, one instance of the querying application receives replies to its query from multiple cores in the DataSpace. The longer query time for the 256 application querying cores is due to load variability and interference in the host system.

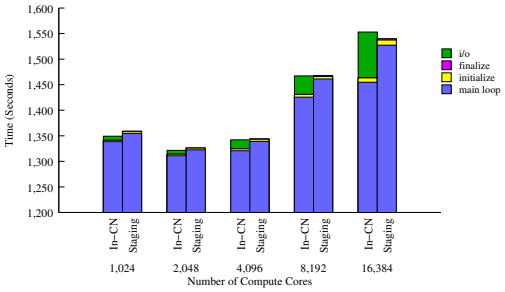
Note that the DataSpaces service indexes particles data and responds to all queries in less than 80 seconds. Considering the 120 second I/O interval, such an online querying service can function effectively and without blocking the simulation.

C. Pixie3D Performance

Pixie3D performance is evaluated on the XT4 partition of Jaguar. Production runs use one MPI process per compute



(a) Pixie3D Simulation Performance and Cost Improvement



(b) Pixie3D Total Execution Time Breakdown

Figure 10. Pixie3D Simulation Performance

core. The data output from each process mainly consists of eight double-valued arrays. Each local array is part of a 3D global array, respectively. Our tests use a 32x32x32 local array size, which is a typical setting for production runs. For each run, the simulation performs I/O about every 100 seconds. The ratio of compute cores to staging cores is maintained at 128:1 during weak scaling. Each process generates about 2 MB of data making this ratio workable.

Pixie3D is tested with an In-Compute-Node configuration and a Staging configuration. For the In-Compute-Node configuration, each MPI process writes output data to a single BP file using the ADIOS synchronous MPI-IO method. This results in a file in which local array chunks are scattered. In the Staging configuration, output data of compute nodes are sent to the Staging Area where they are merged to form larger, contiguous global arrays.

Fig. 10(b) shows the total simulation execution time for both the In-Compute-Node and Staging configurations. The Staging configuration slows the simulation in most cases by 0.01% to 0.7% compared to the In-Compute-Node configuration. This is because unlike GTC, Pixie3D does not have enough computation intensity for asynchronous I/O to be an effective technique for offloading data. In each iteration, the inner loop of Pixie3d performs collective communications (MPI_Reduce and MPI_Bcast) multiple times, and 'between' the mass communications, there are computations that only last about 0.7 seconds. This makes it difficult to overlap data movement with computation without impacting Pixie3D's intensive messaging, as shown by experimental results demonstrating increases in main loop times due to asynchronous data movement. Although the I/O blocking time is well hidden, since it is such a tiny portion of the total execution time, this savings cannot outweigh the slowdown

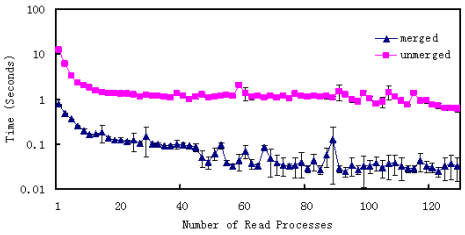


Figure 11. Time to read one global array of one time step from two 80GB BP files. 'merged' denotes the read time from a file written from Staging Area and 'unmerged' denotes the read time from a file written from compute nodes directly. Both files are generated by 4096-compute-core runs.

of computation due to communication interference. The operations tested for the GTC application were all intended to be performed before any data analysis were performed, in order to speed read operations. The same is true for Pixie3D's data reorganization operation. While GTC's operations were a win-win for both writing and reading at all scales, Pixie3D's data reorganization requires larger job sizes to reach a tipping point where simulation performance can be improved by employing staging. Figure 10(a) shows the total cost of CPU seconds. As the simulation scales up, the I/O overhead weighs more in total execution time, and hence the impact of computation caused by data staging becomes less evident. Overall, there is a trend that the cost of the Staging approach catches up with that of the In-Compute-Node approach with increased simulation scale.

It is worth examining the savings generated during reading operations due to the reorganized data. Fig. 11 shows the read performance on two files generated by two 4096-compute-core runs with Staging and In-Compute-Node configurations, respectively. This result, along with the simulation cost shown in Fig. 10(a), shows that at the scale of 4096 compute cores, 0.93% additional cost in simulation yields 10 times improvement in read performance of output data. This saving is more evident as scale increases.

In summary, this section's performance results show that in-transit data manipulation enabled by PreDatA middleware can improve the latency to operation completion compared with the offline approach. It can reduce the overall wall clock time of the simulation at large scales, and it can reduce the impact on the shared file system when compared against both online and offline configurations. It is also shown that higher-level data services can be efficiently built on top of PreDatA middleware.

VI. Related Work

In this section we summarize previous research related to PreDatA work.

Scalable I/O and Data Analytics. Efficient access, understanding and management of voluminous and complex data generated by scientific simulations presents daunting challenges to both computational and computer scientists [19], [36]. Recent work in parallel file systems [8], [35], [48] and I/O middleware [22], [30], [38], [50], [52] aims at optimizing data storage and access for scientific application workloads. Beyond pure high I/O bandwidth, however, scientists also require complex data analysis, search, and visualization technologies to facilitate better understanding of their data. Specialized data preparation, such as sorting, filtering, and indexing, is needed before data can be understood or visualized [9], [39], [43]. Our work extends the I/O middleware stack to exploit computational power along the output data flow to perform data preparation, characterization, and re-organization, which would facilitate subsequent data analysis.

Data Staging and Offloading in supercomputers. Previous work on data staging and asynchronous I/O [4], [16], [24], [25], [32], [34], [41] derives substantial performance advantages from hiding I/O latency with asynchronous data movement. Our recent work [1], [2] shows the importance of minimizing interference of asynchronous data movement with the application to achieve overall improvements in simulation time. One observation is that the computational resources on staging nodes are often under-utilized and the time intervals between I/O dumps are sufficiently large for extra processing on buffered data. In this paper, we take one step forward and demonstrate the use of staging nodes for a diversity of data operations to achieve not only high write performance, but also high read performance and timely monitoring of output data and simulation.

Active Storage. Active Storage [37] deploys data processing operations directly on the storage nodes where the data are buffered to reduce the amount of data movement between storage and compute nodes. The storage nodes have limited computation and memory resources which are shared among applications, so one potential problem with Active Storage is how to manage such resources to meet deadlines for multiple applications and minimize performance downgrade of storage nodes. Abacus [5] demonstrates the benefit of flexible, dynamic function placement in Active Storage, and we are going to investigate similar mechanisms for PreDatA.

In-situ Data Analytics and Visualization. Hercules [46] applies an end-to-end approach to tightly couple all simulation components, including meshing, partitioning, solver, and visualization, and runs all components on the same supercomputer. It eliminates intermediate I/O and data movement between simulation components to address the I/O bottleneck, but requires scaling data analysis and visualization to the level where simulation runs and all simulation components must be changed to efficiently share data with each other. PreDatA couples the Staging Area with the application more loosely and through the ADIOS interface, thereby requiring minimal changes to application code and providing improved flexibility in composing the simulation's output and analysis pipeline.

Scientific Workflows. Scientific workflow systems like Pegasus [14] and Kepler [31] are used to automate scientific data and simulation management. Unlike the end-to-end

approach used in In-situ visualization mentioned above, components in the workflow are usually connected via a file-based interface, so that the performance of the workflow is very sensitive to data placement and movement and is easily affected by poor I/O performance [13]. PreDatA can serve as an early stage in output pipeline to apply application-specific data reduction, validation, and filtering operation before data is moved to disks, to reduce the data volumes to be processed in subsequent workflow steps.

Scientific Data Stream Processing. Scientific data stream processing, such as filtering [6], sampling [47], query [27], and transformation [23] complements our work. This is because PreDatA can be used either as an in-transit data processing framework for implementing streaming processing tasks, or as a data forwarding layer to directly feed data to existing streaming processing systems.

Code Coupling. Memory-to-memory code coupling addresses some of the issues faced by PreDatA, such as data movement and re-distribution [3], [26]. PreDatA provides the underpinnings for supporting the rich model-model communications needed for inter-application interactions [15].

Interactive Computational Steering. Runtime steering can aid scientists in debugging and monitoring their simulations [20], [45]. The capability of extracting and inspecting data from running simulation with small overhead and interference makes PreDatA a potential infrastructure for online steering of running application.

Data-intensive Computing in the HPC Domain. Recently, there is increased interest in building high-level abstractions and programming models for data intensive applications in HPC domain. HiMach [44] applies the MapReduce model to analyze molecular dynamics simulation trajectories and shows some efficiency at tera-byte scale. In contrast, experiences from implementing materialized ground models [40] show poor performance of MapReduce because some of the features provided by MapReduce are unnecessary for its target application. AllPairs [33] gains similar insights in that a mismatch between the application workload and the available MapReduce abstractions can result in poor performance. The two-pass streaming model used by PreDatA appears sufficient for the applications we have used, but it remains an important item of future work to investigating higher level abstractions and a suitable programming model for future PreDatA codes.

VII. Conclusions and Future Work

This paper presents the PreDatA middleware for preparing and characterizing data 'in-transit', that is, while data is being produced by the large scale simulations running on peta-scale machines. PreDatA offloads output data from a running simulation with low-overhead using asynchronous data extraction. It also exploits the computational power of staging nodes residing on the peta-scale machine and associated with each large-scale application to perform select data manipulations. PreDatA enhances the scalability and flexibility of current I/O stacks on HEC platforms and is useful for data pre-processing, runtime data analysis and inspection. The DataSpaces services now being integrated into PreDatA also demonstrates its potential utility for rich model-model interactions in large-scale HPC codes. Performance evaluations with several production scientific applications on ORNL's peta-scale machines show the feasibility of the PreDatA approach as well as the performance advantages derived from using the PreDatA I/O stack compared to existing synchronous approaches.

Several interesting insights distinguish the class of preparatory data analysis provided by PreDatA from other means of running such codes:

Easy data movement. Efficient PreDatA operations leverage the ability of the high performance machines to transfer data to the preparatory processes without causing measurable overheads to the application.

Global data knowledge. The availability of global knowledge is essential in the pre-processing of data for analysis and for application interaction.

Flexible partitioning of the pre-analytics pipeline. The performance requirements for a pre-analytics pipeline require the ability to flexibly partition complex data processing operations.

Streaming computation. The large size of data being processed and the limitation on available memory space within the processing area can limit the scope of viable operations in the processing pipeline. A streaming computational model circumvents these limitations by providing a window on the data in which more expansive pipelines can be utilized.

Standard programming model. Scientific developers are familiar with using standard APIs such as MPI for developing analytical programs. An architecture that seeks to address the needs of the scientific user must be able to utilize standard parallel programs for the pre-analytic data processing pipeline.

Our future work leverages these insights in several ways. First, we plan to define a higher level programming model and abstractions to support a broader set of applications and pre-data analytics, including online data diagnostics and code coupling. Second, we are going to investigate mechanisms for dynamically adapting system configuration and operation placement to cope with changing resource availability or performance characteristics. Third, we will develop performance models for sizing staging areas and provisioning their services.

Acknowledgment

This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05- 00OR22725. This research is based in part upon

work supported by the National Science Foundation through the High- End Computing University Research Activity (HECURA) Grant Number 0621538. This work was also partially supported by The Extreme Scale Systems Center at ORNL and the Department of Defense.

REFERENCES

[1] H. Abbasi, J. Lofstead, F. Zheng, S. Klasky, K. Schwan, and M. Wolf. Extending i/o through high performance data services. In *CLUSTER*, 2009.

[2] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. Datastager: scalable data staging services for petascale applications. In *HPDC*, 2009.

[3] H. Abbasi, M. Wolf, K. Schwan, G. Eisenhauer, and A. Hilton. Xchange: coupling parallel applications in a dynamic environment. In *CLUSTER*, 2004.

[4] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan. Scalable i/o forwarding framework for high-performance computing systems. In *CLUSTER*, 2009.

[5] K. Amiri, D. Petrou, G. R. Ganger, and G. A. Gibson. Dynamic function placement for data-intensive cluster computing. In *USENIX Annual Technical Conference*, 2000.

[6] M. D. Beynon, R. Ferreira, T. M. Kurç, A. Sussman, and J. H. Saltz. Datacutter: Middleware for filtering very large scientific datasets on archival storage systems. In *MSST*, 2000.

[7] R. Brightwell, T. Hudson, K. T. Pedretti, R. Riesen, and K. D. Underwood. Implementation and performance of portals 3.3 on the cray xt3. In *CLUSTER*, 2005.

[8] P. Carns, S. Lang, R. Ross, M. Vilayannur, J. Kunkel, and T. Ludwig. Small-file access in parallel file systems. In *IPDPS*, 2009.

[9] S. Center. Scidac scientific data management center. https://sdm.lbl.gov/sdmcenter/, September 2009.

[10] L. Chacón. A non-staggered, conservative, $\nabla \dot{s} B \rightarrow = 0$, finite-volume scheme for 3D implicit extended magnetohydrodynamics in curvilinear geometries. *Computer Physics Communications*, 163:143–171, Nov. 2004.

[11] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. Technical Report UCB/EECS-2009-136, EECS Department, University of California, Berkeley, Oct 2009.

[12] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI*, 2004.

[13] E. Deelman and A. Chervenak. Data management challenges of data-intensive scientific workflows. In *CCGRID*, 2008.

[14] E. Deelman, G. Singh, M. hui Su, J. Blythe, A. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13:219–237, 2005.

[15] C. Docan, M. Parashar, J. Cummings, N. Podhorszki, and S. Klasky. Experiments with Memory-to-Memory Coupling for End-to-End Fusion Simulation Workflows. Technical Report TR-104, Center for Autonomic Computing (CAC), Rutgers University, July 2009.

[16] C. Docan, M. Parashar, and S. Klasky. Dart: a substrate for high speed asynchronous data io. In *HPDC*, 2008.

[17] G. Eisenhauer. Evpath: event transport middleware layer. http://www.cc.gatech.edu/systems/projects/EVPath/, September 2009.

[18] G. Eisenhauer, F. E. Bustamante, and K. Schwan. Native data representation: An efficient wire format for high-performance distributed computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(12):1234–1246, 2002.

[19] J. Gray, D. T. Liu, M. Nieto-Santisteban, A. Szalay, D. J. DeWitt, and G. Heber. Scientific data management in the coming decade. *SIGMOD Rec.*, 34(4):34–41, 2005.

[20] W. Gu, G. Eisenhauer, K. Schwan, and J. S. Vetter. Falcon: On-line monitoring for steering parallel programs. *Concurrency - Practice and Experience*, 10(9):699–736, 1998.

[21] C. Jones, K.-L. Ma, A. Sanderson, and L. R. M. Jr. Visual interrogation of gyrokinetic particle simulations. *J. Phys.: Conf. Ser.*, 78(012033):6, 2007.

[22] W. keng Liao and A. N. Choudhary. Dynamically adapting file domain partitioning methods for collective i/o based on underlying parallel file system locking protocols. In *SC*, 2008.

[23] S. Klasky, S. Ethier, Z. Lin, K. Martins, D. McCune, and R. Samtaney. Grid -based parallel data streaming implemented for the gyrokinetic toroidal code. In *SC*, 2003.

[24] J. Lee, R. B. Ross, S. Atchley, M. Beck, and R. Thakur. Mpi-io/l: efficient remote i/o for mpi-io via logistical networking. In *IPDPS*, 2006.

[25] J. Lee, R. B. Ross, R. Thakur, X. Ma, and M. Winslett. Rfs: efficient and flexible remote file access for mpi-io. In *CLUSTER*, 2004.

[26] J.-Y. Lee and A. Sussman. High performance communication between parallel programs. In *IPDPS*, 2005.

[27] Y. Liu, N. Vijayakumar, and B. Plale. Stream processing in data-driven computational science. In *GRID*, 2006.

[28] J. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin. Flexible io and integration for scientific codes through the adaptable io system (adios). In *CLADE at HPDC*, 2008.

[29] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Input/output apis and data organization for high performance scientific computing. In *PDSW at SC*, 2008.

[30] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Adaptable, metadata rich io methods for portable high performance io. In *In IPDPS*, 2009.

[31] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[32] X. Ma, J. Lee, and M. Winslett. High-level buffering for hiding periodic output cost in scientific simulations. *IEEE Trans. Parallel Distrib. Syst.*, 17(3):193–204, 2006.

[33] C. Moretti, J. Bulosan, D. Thain, and P. J. Flynn. All-pairs: An abstraction for data-intensive cloud computing. In *IPDPS*, 2008.

[34] A. Nisar, W. keng Liao, and A. N. Choudhary. Scaling parallel i/o performance through i/o delegate and caching system. In *SC*, 2008.

[35] R. Oldfield, L. Ward, R. Riesen, A. B. Maccabe, P. Widener, and T. Kordenbrock. Lightweight i/o for scientific applications. In *CLUSTER*, 2006.

[36] PDSI. Scidac petascale data storage institute. http://www.pdsi-scidac.org/, September 2009.

[37] J. Piernas, J. Nieplocha, and E. J. Felix. Evaluation of active storage strategies for the lustre parallel file system. In *SC*, 2007.

[38] M. Polte, J. Simsa, W. Tantisiriroj, and G. Gibson. Fast log-based concurrent writing of checkpoints. In *PDSW at SC*, 2008.

[39] O. Rübel, Prabhat, K. Wu, H. Childs, J. Meredith, C. G. R. Geddes, E. Cormier-Michel, S. Ahern, G. H. Weber, P. Messmer, H. Hagen, B. Hamann, and E. W. Bethel. High performance multivariate visual data exploration for extremely large data. In *SC*, 2008.

[40] S. W. Schlosser, M. P. Ryan, R. Taborda-Rios, J. López, D. R. O'Hallaron, and J. Bielak. Materialized community ground models for large-scale earthquake simulation. In *SC*, 2008.

[41] K. E. Seamons, Y. Chen, P. Jones, J. Jozwiak, and M. Winslett. Server-directed collective i/o in panda. In *SC*, 1995.

[42] R. R. Sinha and M. Winslett. Multi-resolution bitmap indexes for scientific data. *ACM Trans. Database Syst.*, 32(3):16, 2007.

[43] K. Stockinger, J. Shalf, E. W. Bethel, and K. Wu. Dex: Increasing the capability of scientific data analysis pipelines by using efficient bitmap indices to accelerate scientific visualization. In *SSDBM*, 2005.

[44] T. Tu, C. A. Rendleman, D. W. Borhani, R. O. Dror, J. Gullingsrud, M. Ø. Jensen, J. L. Klepeis, P. Maragakis, P. Miller, K. A. Stafford, and D. E. Shaw. A scalable parallel framework for analyzing terascale molecular dynamics simulation trajectories. In *SC*, 2008.

[45] T. Tu, H. Yu, J. Bielak, O. Ghattas, J. C. López, K.-L. Ma, D. R. O'Hallaron, L. Ramirez-Guzman, N. Stone, R. Taborda-Rios, and J. Urbanic. Analytics challenge - remote runtime steering of integrated terascale simulation and visualization. In *SC*, 2006.

[46] T. Tu, H. Yu, L. Ramirez-Guzman, J. Bielak, O. Ghattas, K.-L. Ma, and D. R. O'Hallaron. Scalable systems software - from mesh generation to scientific visualization: an end-to-end approach to parallel supercomputing. In *SC*, 2006.

[47] H. Wang, S. Parthasarathy, A. Ghoting, S. Tatikonda, G. Buehrer, T. M. Kurç, and J. H. Saltz. Design of a next generation sampling service for large scale data analysis applications. In *ICS*, 2005.

[48] B. Welch, M. Unangst, Z. Abbasi, G. A. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou. Scalable performance of the panasas parallel file system. In *FAST*, 2008.

[49] H. Yu and K.-L. Ma. A study of i/o methods for parallel visualization of large-scale data. *Parallel Comput.*, 31(2):167–183, 2005.

[50] W. Yu, J. S. Vetter, and S. Oral. Performance characterization and optimization of parallel i/o on the cray xt. In *IPDPS*, 2008.

[51] L. Zhang and M. Parashar. Seine: a dynamic geometry-based shared-space interaction framework for parallel scientific applications. *Concurrency and Computation: Practice and Experience*, 18(15):1951–1973, 2006.

[52] X. Zhang, S. Jiang, and K. Davis. Making resonance a common case: A high-performance implementation of collective i/o on parallel file systems. In *IPDPS*, 2009.

[53] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, Q. Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. Predata - preparatory data analytics on peta-scale machines. Technical Report GIT-CERCS-10-01, Georgia Institute of Technology, January 2010.