I/O Containers: Managing the Data Analytics and Visualization Pipelines of High End Codes

Jai Dayal, Jianting Cao, Greg Eisenhauer, Karsten Schwan, Matthew Wolf, Fang Zheng
Georgia Institute of Technology
Hasan Abbasi, Scott Klasky, Norbert Podhorszki
Oak Ridge National Labs
Jay Lofstead
Sandia National Labs

***Abstract*—Lack of I/O scalability is known to cause measurable slowdowns for large-scale scientific applications running on high end machines. This is prompting researchers to devise 'I/O staging' methods in which outputs are processed via online analysis and visualization methods to support desired science outcomes. Organized as online workflows and carried out in I/O pipelines, these analysis components run concurrently with science simulations, often using a smaller set of nodes on the high end machine termed 'staging areas'.**

This paper presents a new approach to dealing with several challenges arising for such online analytics, including: how to efficiently run multiple analytics components on staging area resources providing them with the levels of end-to-end performance they need and how to manage staging resources when analytics actions change due to user or data-dependent behavior. Our approach designs and implements middleware constructs that delineate and manage I/O pipeline resources called 'I/O Containers'. Experimental evaluations of containers with realistic scientific applications demonstrate the feasibility and utility of the approach.

***Keywords*-Data Staging, Data Analytics, in-Situ, Visualization, Scalable I/O, Runtime Management, resource sharing**

I. Introduction

On current generation petascale platforms, scientific applications like the GTC [1] and GTS [2] fusion simulations are already generating terabytes of data every few minutes. To store these immense data volumes without overwhelming the file systems attached to petascale machines, developers have turned to machine-resident methods for 'in situ' data processing and I/O data staging [3], [4], [5], [6], [7]. With such methods, data can be reduced or compressed before it is placed into storage [8], [9], buffered and better organized for efficient data movement [3] and storage on parallel file systems [10] or for subsequent (post-storage) access and manipulation, and more generally, pre-processed or re-organized in the ways needed for data analytics or visualization [11], [12].

While the above methods and infrastructures have become key to the I/O processes running on high end machines, as we move to the exascale, online analytics make possible new ways to better understand and control scientific simulations while they run. This includes (i) continuously ascertaining simulation validity, permitting it to be terminated or corrected without undue waste of machine resources, (ii) gaining rapid insights into the scientific processes being simulated (online visualization), or even (iii) enabling methods for application steering. The result of these developments, however, is that the new 'normal' for high end codes is that they will be structured as parallel simulations running jointly with a dynamic set of tightly coupled parallel analytics, visualization, and validation codes. This combination of analysis components deployed along the I/O path are called an *I/O pipeline*. A consequent challenge, then, is *how to manage the machine resources used by the ensemble for optimal time to solution?*

Online management of I/O pipelines is important for several reasons. First, during an application run, its I/O characteristics and analytics requirements can vary substantially ('normal' operation vs. checkpointing). Second, there will be analytics that prove too expensive to run online perhaps due to computational or memory requirements and must be moved offline to avoid blocking the simulation. Third and more generally, the throughput and scaling characteristics of analytics computations can be dynamic and they may not be well-understood as such components are often developed separately from the core simulations. Important features of the management system, therefore, are support for continuous online profiling and monitoring with fine-grained launch capabilities. Such performance information, then, can be used to inform both users and resource management methods about opportunities and limitations in their execution environment.

This paper describes the *I/O container* approach, depicted in Fig. 1, to managing dynamic I/O and analytics pipelines on high end machines. Containers provide a common, controllable execution framework for the diverse analytics components running in conjunction with a scientific simulation. Such components can be compiled and deployed separately each in their own container, have well defined inputs and outputs [13], can be parallel (MPI or threads), and may have dependencies across them.

For this model, I/O containers offer:

1) *controlled resource usage:* a container provides and manages resources for the analytics component

mapped to it;

2) *per component management:* a container offers to its component an actively managed execution environment that ensures isolation from other containers' resource demands.
3) *metric-driven operation:* beyond the per-container methods for managing its resources, the container runtime can also enforce global goals, driven by metrics of interest to end users, such as priorities or performance requirements. Enforcing such properties will require operations such as changing resource allocations amongst containers.

Finally, since the control and management actions taken by container software must not place applications or analytics components into inconsistent states or cause resource loss due to failure, we are experimenting with fault-resilient transactional techniques [14] that can make strong guarantees about the successful completion of certain control actions. An example is a guarantee that a resource removed from one container is successfully given to another.

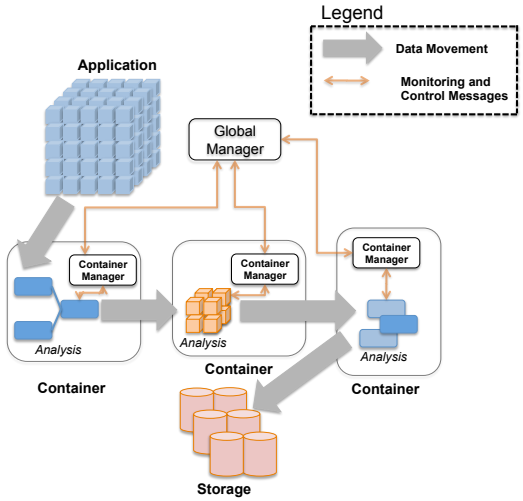

Figure 1: I/O Pipeline Showing I/O Containers

Using I/O containers permits end users to focus on analytics functionality and correctness instead of having to adjust and 'size' component resource allocations and/or tune their performance for online use. Using containers, a user can also launch a visualization code when needed. Further, containers make viable automated methods for resource allocation and re-allocation that operate in response to components' changes in resource needs. This is demonstrated in our current implementation where multiple containers segment the common staging area used to execute online analytics for a scientific simulation. This is done by running online I/O data visualization with, for example, ParaView in one container while running analytics using VTK in another container. In this scenario, a dynamic requirement for additional resources to run the analytics can be met by 'stealing' resources from the visualization container, if it does not need them, or by using spare staging resources, if available. All such actions are enabled by the container framework via its well-defined component interfaces and flexible inter-component communication methods and they are supported with online monitoring that profiles individual containers for their current resource usage. We also use such monitoring to drive bottleneck detection to determine the end-to-end latencies containers, arranged as processing pipelines, experience to avoid blocking the pipeline.

Current I/O staging technologies do not yet offer functionality that dynamically manages the tightly coupled analytics run with high end codes. In our own earlier work, statically profiled analysis routines are run in static configurations sized to be resource-rich for worst case data volumes and processing needs [11].

This paper demonstrates the concept and an implementation of I/O containers on a Cray Portals cluster. Supporting the LAMMPS molecular dynamics application, representative analytics and visualization components serve the purpose of discovering and dealing with 'crack formation' in the substrate being modeled, as explained in more detail in [15]. Each of the analytics components have different scalability characteristics and they can utilize different degrees of parallelism for faster (or slower) execution in response to changes in I/O behavior and requirements. Further, some of the analytics methods are sufficiently computationally expensive that running the components using them is really only merited when some interesting application-level event, such as when the formation of an inelastic deformation has occurred. Ideally, the system should dynamically respond to changes due to the data itself (i.e., when a crack is detected) as well as due to online user direction (i.e., add this filter now while I'm looking at the output). In all such cases, container management can dynamically adjust the resources allocated to balance component pipeline execution.

Our initial containers implementation is based upon the EVPath and DataTap staging solutions [16], [3] and experimental evaluations use analytics components derived from those described for the Smartpointer software [15]. Current work experiments with the S3D [17] combustion modeling code and the numerous analysis and visualization components developed for it to perform flame front tracking and visualization.

Future work with containers will also apply the concept and the control transactions mentioned above to the CTH [18] shock physics code as part of a data pipeline that turns the raw atomic data into materials fragments to allow tracking. By moving this workflow online, data can be staged and processed, both generating fragments and tracking them as they evolve in the simulation, opening new opportunities for understanding the physics at work.

The implementation of I/O containers evaluated in this paper demonstrates the utility of the containers approach with ongoing implementation hardening it and experiment-

ing with the additional use cases mentioned above. The performance results shown in Section IV show containers do not add notable overheads to running the components they manage. More importantly, they show containers to be potentially useful. Specifically, when using a simple management policy, containers improve the end-to-end latency through an analysis pipeline and prevent application blocking by taking unneeded components offline.

II. Related Work

A number of on-going projects could benefit from I/O containers. In particular, while currently realized for ADIOS and analytics pipelines, containers are equally useful to other 'data staging' solutions as long as they describe and use well-defined component interfaces. However, without using actual virtualization solutions like Palacios [19], additional programming and integration efforts will be required to add the concept to tightly integrated analytics codes in which individual actions are not separately defined and/or use well-defined component APIs, such as IBM's System S [20].

The basic idea of containers presented here is akin to the 'island' or 'container' concepts developed for both multicore/SMP platforms and datacenter systems. Concerning the former, containers are similar in spirit to the hypervisor-level 'resource islands' [21] in which per island resource managers run entirely different scheduling algorithms, but cross-island cooperation (or global control) is needed to ensure platform or application properties spanning multiple islands. The container implementation shown in this paper is entirely in middleware and targets HPC clusters. Mesos' [22] original intent was to permit fine grain resource sharing across multiple applications running in datacenter systems with only recent work (unpublished) exploring resource trading methods that may also be suitable for the HPC domain.

Readers may correctly recognize that the basic idea of 'containers' relates to research conducted under the guise of virtualization, such as projects like [23], [24], [19]. In this context, a container can be viewed as a lightweight construct similar to a hypervisor for a 'virtual machine' that explicitly manages the execution environment of parallel components. Containers, however, only adopt some of the functionality of modern hypervisors and they are implemented as user-space abstractions for two reasons. First, this makes them suitable for current and next generation high end machines. Second, as opposed to closed hypervisors, we permit end users to realize their own custom management policies for their applications.

Our own earlier work on 'service augmentation' [25] demonstrates the utility of attaching Quality of Service (QoS) management actions to I/O pipelines and shows that container principles can be applied to other data staging or streaming infrastructures and systems, including systems like DataSpaces [6] and Glean [7], both of which use componentized approaches. In contrast, it would be more difficult to apply containers to the graph-structured, fine-grained stream processing actions run by infrastructures like System S or StreamIT [20], [26]. Our approach operates at the coarser-grain level of the typical analytics actions carried out on high end machines often implemented as parallel MPI codes spanning many machine nodes.

III. I/O Containers

I/O containers are run-time abstractions that allow in situ or in transit data processing actions to be embedded into a dynamically managed execution environment. Each single container manages an executable that carries out analytics tasks on the data it ingests, with some examples mentioned in Section I.

More complex structures, like pipelined components, are supported by chains of containers supervised by a higher level manager interfacing to per-component manangers. In all such cases, components are run on the machine resources made available by the container and controlled by potentially container-specific management and scheduling.

Fig. 1 depicts a conceptual model of I/O containers. A multi-level management scheme can maintain both container-level and global (i.e., across all containers) properties. Such distributed management is supported with a flexible monitoring and control infrastructure gathering needed information and then issuing appropriate control operations. Controlled data movements avoid blocking a sender by a receiver that is not well prepared to accept new data.

A. I/O Container Runtime Constraints and Desired Properties

In situ I/O data processing can be performed in multiple locations: compute nodes, staging nodes, or an ancillary analysis/visualization cluster. Our current implementation uses staging nodes in a batch scheduler environment, and as a result we must realize the desired properties under limited staging resources.

Obtaining desired performance with limited staging resources. With current high end machines, batch schedulers assign to each user some number of requested nodes for the entire duration of their application's execution and the user must determine how to partition these nodes. Typical relative sizes of staging to simulation nodes in our past experiments on the Jaguar petascale machine range from 1:512 to 1:2048 [11], [3]. These ratios impose stringent resource constraints on staging area computations and on the volumes of data stored on staging nodes.

From a user's perspective, certain analytics may be important or critical to complete whereas others (e.g., visualization) can perhaps be delayed. Examples of critical analytics are those that affect how output is performed or even steer or control the simulation itself. Substantial recent work, for instance, is investigating continuous methods validating that

simulations continue to produce scientifically meaningful results, as with uncertainty quantification.

As a demonstration of such systems, the example used in this paper is analytics used for the online detection of crack formation in a material being modeled by the LAMMPS molecular dynamics code. Using this as an illustrative example, one can follow several implications for embedding analytics into I/O containers:
(i) *dynamic response* – since analytics workloads can change during a program's run (e.g., when crack formation imposes sudden loads on analytics or visualization actions), analytics components' resource needs may change, e.g., to run sufficiently fast to prevent application blocking;
(ii) *isolation* – new analytics may be launched in response to interesting program events, but this must not jeopardize the execution of other analytics actions required by the application; and
(iii) *metric-driven operation under resource limits* – additional workload and new analytics actions must be carried out in the confines of the staging resources allocated when an application run starts with the performance needed by said actions and guided by metrics that may differ across components and/or use cases. Consider, for instance, a container running data aggregation (e.g. scientific checkpointing) code: it need not complete writing data to stable storage until the next timestep arrives. This is in contrast with another container running code for crack discovery: it should complete with low latency to minimize the danger of wasted CPU cycles on scientifically invalid computation.

We conclude from these facts that (1) *I/O containers must be actively and continuously managed* in ways determined by their functionality and properties and by user requirements. Concerning properties, some components may scale well when they are given additional nodes (e.g., linear speedup) while others may not. If sufficient resources are available, this may be controllable by spawning multiple component instances fed by subsequent simulation output steps while with other components, data may be filtered to reduce component workload. In general, these comments illustrate that there must be diversity in the management methods used with containers. This prompts us to design a container framework in which (2) *I/O containers support per component custom management with global management adaptable to current usage or needs*.

Componentized operation. With I/O containers, each analytics action runs as a separate application (i.e., component), with well-defined input and output interfaces. This makes it possible to run entirely different, dynamically swappable analytics codes without requiring them to be integrated into a single executable. Reusability of old container actions can be maintained while allowing easy evolution of new actions. In our implementation, we utilize the ADIOS read/write interfaces to define the input and output for our actions. This gives us all of ADIOS's tools for easing the user specification of component interfaces, but at this time, ADIOS lacks support and programming model for describing multi-component interactions.

These facts impose limitations on how to manage containers as decisions made upstream, for instance, have a direct effect on downstream operation and they directly affect the end-to-end QoS properties of I/O pipelines [27], [25]. An extreme example is the removal of a component from the I/O pipeline rendering its downstream components useless. Stated differently, there will typically be well-defined Service Level Agreements (SLAs) for running analytics actions on the outputs produced by simulations. The simplest case is the one mentioned earlier where analytics must complete before the application initiates its next output step so as to prevent blocking on I/O. Other SLAs include limits on the end-to-end latency through an I/O pipeline (e.g., the analytics results affect subsequent steps or even steer the application itself) and reliability requirements in terms of storage actions having completed to avoid losing checkpoint data. For I/O containers, this results in the additional requirements that (3) *management actions are guided by user-determined metrics driving per-container and cross-container (i.e., global) management policies*.

Flexible monitoring and control. Active management of varied components with potentially custom policies cannot be carried out unless (4) *there is information about the current state of resource availability, usage requirements, and component behavior*. The I/O container framework satisfies these needs by use of built-in monitoring primitives, the execution of which is triggered by per component (i.e., per container) or global events. The data captured by monitoring can be varied and so can the ways in which monitoring data is processed to detect events of interest to container control. The example shown in this paper tracks of end-to-end latency of actions in a pipeline-structured set of containers, with control actions taken when the latency SLA specified by the user is exceeded.

Reliable control. Care must be taken to apply agreed-upon consistency for distributed infrastructure control such as is discussed here. For instance, when two containers trade resources, the resource is first removed from the donor and then added to the recipient. Failures incurred during such trades can lead to inconsistent system states in which one container believes a resource was removed, but the second never completed the action that added it to its resource inventory. To address such issues, we are experimenting with (5) *control transactions that rule out such situations under arbitrary failure conditions*.

In summary, the I/O container framework developed in our work can be used to realize (1) per-container and (2) global management policies, (3) customized to their current use and to meet user requirements, (4) enabled by online monitoring of the varied metrics of relevance to different policies, and (5) made resilient to failure via control transactions.

B. Software Architecture and Implementation

Figure 1 depicts a high-level view of the I/O container software architecture. Each container is shown to have its local manager responsible for the control and monitoring actions associated with that container. The container manager also serves as the point of contact for a container's interactions with global management, which in our current implementation is a distinct global manager responsible for maintaining naming data, monitoring, and enforcing global goals (e.g., SLAs like those limiting end-to-end latencies across several components). Global management could also be implemented with peer-to-peer algorithms that operate across local managers, but at the scales of our current system, it is preferable to use a single global management entity. Methods like those described in ZooKeeper [28] can be used to maintain high levels of resilience for this potential single point of failure.

Each local manager has complete control over its interactions with the cohort of actions it manages. This is important because only the local container manager understands the characteristics of its designated action(s). By decoupling management in this fashion, it becomes possible to customize the management plane and simplify global management as it is not required to understand all computational models, I/O characteristics, QoS requirements, etc.

The monitoring shown in Figure 1 provides managers the information needed for making management decisions. For example, the global manager might observe a slowdown in one container that can be remedied by adding resources to it. In our design, the global manager has an aggregate view of the environment so that it can use monitoring for bottleneck analysis, for instance. The lower level managers, then, are the ones that directly interact with their component instances to gauge current throughput, queue lengths, and estimates of how resource additions or removals would impact them. Based on such information, they can assess how many resources are needed to remedy bottlenecks, for instance, and it is this multi-segment distributed management that allows for overall SLA satisfaction.

Containers control data movement so as to not overwhelm receivers with data they cannot handle and in order to better utilize the machine's interconnect (e.g., through communication scheduling, as described in DataStager [3]). Further, container managers communicate for monitoring and control purposes and they must understand inter-component communications. They must know from where data is received and where to send outputs after completing components' analytics actions.

1) Container Implementation: A representative use case illustrates the implementation of I/O containers based on the widely popular LAMMPS (Large Scale Atomic/Molecular Massively Parallel Simulator (LAMMPS)) code. It is written with MPI and performs force and energy calculations on discrete atomic particles. After a number of user-defined epochs, the simulation data is output. Depending on the number of particles, the amount of data being output can be from a few megabytes to terabytes in size. Additionally, for fault tolerance reasons, LAMMPS may perform checkpoint operations to periodically save the simulation state.

Table I: Characteristics for SmartPointer Analysis Actions

	Complexity	Compute Model	Dynamic Branching
Helper	$O(n)$	Tree	No
Bonds	$O(n^2)$	Serial, RR, Parallel	Yes
CSym	$O(n)$	Serial, RR	No
CNA	$O(n^3)$	Serial, RR	No

We use the SmartPointer [15] package as an analytics toolkit. It ingests molecular data from the LAMMPS simulation and performs a series of analyses on the data to produce useful data annotations and provide science users with rapid insights like those needed to check the 'health' and scientific state of their running simulations. With SmartPointer, such insights can be gathered both online and as post-processing after data has been moved to disk. The SmartPointer toolkit and the computational pipelines running in the staging area are comprised of a set of codes with a wide range of requirements and expectations, such as different computational models, different runtime complexities, and dynamic code branching. Table I summarizes these characteristics.

The SmartPointer components used in this example are as follows. LAMMPS Helper is an aggregation tree that accepts atomic bonds data from the parallel LAMMPS simulation. The depth and number of nodes in the tree depends on factors such as how much data can fit into memory, how many functions will need this data, and whether the subsequent functions are parallel. Bonds reads atomic data from LAMMPS Helper and determines whether two atoms are currently bonded. It outputs two data types: the atomic data it ingests and an adjacency list representing the bonded atoms. CSym is a central symmetry calculation that is one way to determine whether a bond between atoms, as determined by Bonds, has been broken. CSym reads atomic data and also needs one reference atomic adjacency data set from Bonds. If a break is detected, Bonds then kills itself and notifies the next stage, CNA, to start reading data from Bonds. This scenario represents a dynamic branch in the pipeline. CNA, or Common Neighbor Analysis, is used to do extensive structural labeling of the atomic environment including detecting crystals, faces, and orientation. After this stage, the data is written to disk for this exercise, although in the more general scenario, it might also be sent to online visualization tools for dynamic interaction.

Fig. 2 depicts the I/O containers software stack. At the first level beneath the application, we have the standard ADIOS interface providing us a common API to use a variety of I/O methods. In our current implementation, we use the DataTap library to provide us with controlled data movement between applications and components and

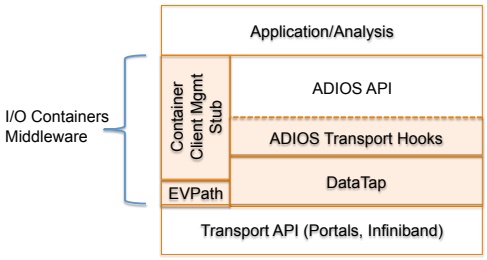

Figure 2: The I/O Containers Implementation Design

employ a thin layer of hooks to integrate DataTap into the ADIOS interface. These layers are enhanced with a container management and control layer based on the EVPath library. This layer is responsible for interacting with the Container Manager (depicted in Fig. 1) and responding to control messages by altering the appropriate state in its neighboring layers.

C. Controlled Data Movement

Data movement between I/O containers uses asynchronous I/O as it has been shown to provide significant improvements in I/O performance with gains up to a factor of 2 [29]. Specifically, the container framework uses the 'DataStager' [3] transport method available through ADIOS. This implementation works as follows: the source container stores data in a buffer, pushes metadata to the target container that then pulls the data when it is ready to do so using the RDMA-based interconnects dominant on high end machines. An added benefit of DataStager is that such pulls can be scheduled to help prevent contention on the HPC interconnect causing the application to run more slowly than using synchronous I/O.

D. Management Framework

An important attribute of I/O containers is the presence of both local – per container – and global managers. Local managers provide the interface to each container's component and the way it can be changed to the global management entity responsible for maintaining multi-container properties and SLAs. A local container manager, for instance, would be aware of the performance improvement (or reduction) seen when adding (or removing) resources from it. A typical example is its knowledge about a parallel component's speedup properties, which could be the result of pre-supplied performance data or directly measured at runtime. The local manager also understands how to capture per-component monitoring data and how to interpret it, such as measuring the data volume and end-to-end delay through the container to diagnose its throughput. Finally, the local managers inform global management about local needs and behavior in support of global goals.

A specific example presented and experimentally evaluated in Section IV is that the higher level authority may detect that a certain container is a bottleneck in a pipeline of containers. The higher-level manager should be able to ask the container-local authority what is needed in order to speed it up (e.g., more cores or more I/O bandwidth). Based on this information and on its understanding of the global environment, the higher-level authority can then make a decision on how to proceed. For example it can shrink one container to increase another or lower the output frequency of one to free up I/O bandwidth for others. Additionally, with this framework, we can allow for some control features that will change the data flow for a container, for example, being able to add hashes of the data to the output for soft error detection.

We currently use the following simple management interfaces between container managers: (i) increase or (ii) decrease a container (by some amount), and (iii) take it offline. When invoked, the functions execute a protocol involving rounds of control messages between the global manager, local managers, and application executables. The EVPath event library [16] is used to communicate these control messages. EVPath allows for the creation of the overlays typically found in monitoring infrastructures. In future work, we are adding the transactional resilience support evaluated as standalone code in Section IV, and in addition, we will support interactive environments in which containers and their analytics components can be launched in mid-run.

It is useful to further elaborate on container resource increases and decreases. Increasing a container's size means that the container will occupy more cores or nodes. How these nodes or cores are used depends on the container's component and its implementation. For example, for a round-robin computational model, we can simply spawn additional parallel instances of the component. For a parallel component relying on MPI, however, increasing the container size would require its complete teardown and restarting a new instance with an increased number of MPI ranks. This is in part due to the basic nature of MPI. However, there are additional complications due to the interaction with the batch scheduler environments on high end machines. For example, on the Cray platforms we use for evaluation, the MPI launch function 'aprun' has the limitation that it is not possible to coalesce applications on the same node that were launched from separate aprun commands. Programming models like those proposed by Charm++, HabaneroC, and the EVPath data streaming infrastructure used in our own work on online data analytics [30], [31], [16] do not suffer from these limitations.

Regardless of such implementation issues, there will be some basic overheads of container resizing that arise from the control protocol being used. To illustrate, Figure 3 depicts an example of the 'increase' protocol and the rounds of control messages among the global manager, container manager, and the component executables. The details of the protocol go beyond the scope of this paper, but in summary, the global manager asks the container to increase its size

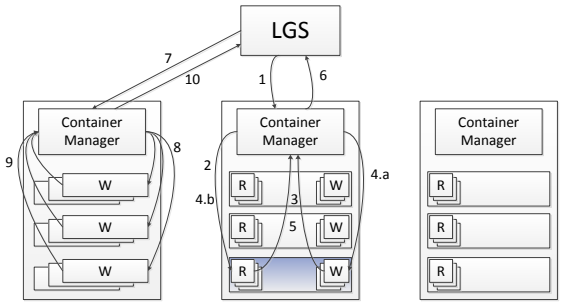

Figure 3: Increase Container Protocol

by some number of nodes. The rounds of messages serve to distribute end-point contact information and serve as notification that certain actions have started or completed. For the 'decrease' operations, the local manager also sends a control message to its components asking them to pause their DataTap writers in order to preserve correctness and avoid loss of time steps while the downsteam container reduces. Experimental evaluations in Section IV demonstrate that these protocol requirements are possible without unduly disturbing the I/O performed by a large-scale simulation.

Since staging area resources are limited, in order to increase a container, we may first need to 'steal' resources from another. The control protocol is similar to what is depicted in Figure 3 except that a container is asked to reduce its node footprint. It may also be that there are no resources available to be traded across containers, and in that case, the only remaining option is to take some non-essential container 'offline'. Container management implements this option so as to guarantee that the stored data will be labeled with its data processing provenance. This makes it possible to keep track of which analytic operations have been performed on the data and which operations need to be performed in the future.

Moving a container offline also requires taking offline all subsequent containers that depend on it. This information is given to the global manager through a configuration file and the global manager is responsible for properly carrying out the necessary control actions. Our current implementation simply has the global manager decreasing each affected container's resources to '0'. When that has been completed, each component replica in the upstream container (still online) has to switch its output method within ADIOS to write to disk using the attribute system to mark the provenance as mentioned before.

E. Container Monitoring

Online monitoring provides the different levels of management with the information needed for tasks like bottleneck detection and container health assessment. Leveraging our previous work on monitoring [32], we implement lightweight monitoring methods that create 'dynamic overlays' across the many nodes on the high end machine participating in the I/O pipeline. We can vary (i) which metrics are captured at the container boundaries of importance to management and inside containers (as determined necessary by local managers), (ii) how often they are captured, and (iii) how they are processed and where such processing is done. As explained in [32], such flexibility in monitoring is important in order to minimize perturbation to applications from the monitoring carried out by I/O containers.

This paper uses online container monitoring to detect bottleneck components in I/O pipelines, which we determine by finding the pipeline's container with the longest average latency. Latency is measured from the time the input data from a timestep enters the component until it exits. We use the EVPath library to transfer the monitoring messages.

IV. Performance Evaluation

The first set of measurements are micro-benchmarks that outline the costs associated with the management operations. After assessing these base costs, we next run experiments that show the usefulness of the container approach. Using LAMMPS and SmartPointer, we demonstrate intra-container improvements through latency management actions by taken by the container manager. We then evaluate end-to-end improvements gained from global management of output pipelines comprised of multiple components and their respective containers. Specifically, we show the time for each timestep to move through the pipeline drastically decreases.

A. Experimental Setup

The core I/O container experiments are run on NERSC's Franklin machine. It is a 9,572 node Cray XT4 machine with quad core AMD 'Budapest' 2.3 Ghz processors, a Portals Network, 38,288 total cores, 78TB of memory, and a peak performance of 352 TFlops. We evaluate the container implementation outlined in Section III-B1.

Our ongoing work investigating transactional techniques for resilience in management operations is being conducted on Sandia's RedSky machine. Redsky is a capacity cluster with a peak performance of 433.5 Tflops. It is a Sun Blade center with Sun X6275 blades containing 2823 nodes running Intel Xeon 5570 processors (8 cores each) with 12 GB of RAM per node and QDR InfiniBand arranged in a 3-D toroidal mesh as the communication fabric. The prototype transaction implementation uses two communication APIs: Open MPI and Sandia's NSSI [33].

B. Results

1) Protocol Overhead: Figures 4, 5, and 6 shows the overheads associated with increase and decrease control operations. Each consist of several rounds of messages, as mentioned in Section III-D. Fig. 4 show the overheads of an increase operation. The x axis represents the size of the increase (increasing a container by 16 replicas, for example). As the graph shows, the communication within a

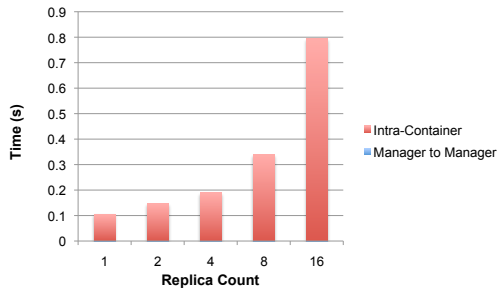

Figure 4: Time to Increase Container Size

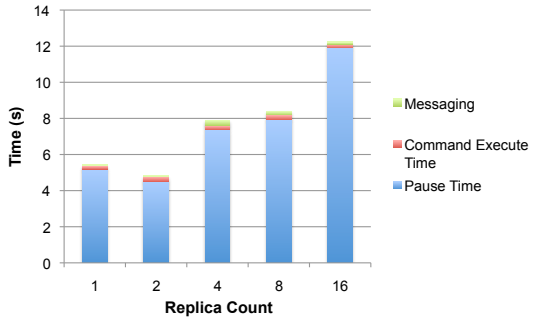

Figure 5: Time to Decrease Container Size

container during a resize is the dominant factor due to the metadata exchanges needed to establish communication with the new replicas. Overheads from point-to-point messages between container managers and the global manager are nearly negligible.

The current implementation of 'increase' is forced to use the 'aprun' command commonly found on Cray high-end machines. That cost has been factored out because it is not inherent to how container management is done, rather, it is an artifact of the particular OS 'batch style' scheduling. The cost of 'aprun' is well known and varies drastically; in our experiments we witnessed 'aprun' times between 3 to 27 seconds, completely drawfing all other measurement in these microbenchmarks.

Fig. 5 shows the overheads associated with decreasing a container's resource footprint. In this particular case, it involves the removal of some round-robin replicas. When doing so, the largest source of overhead is waiting for the replicas' upstream DataTap writers to pause to avoid data loss. This pause has little impact on data flow through the pipeline, however, as DataTap allows for asynchronous writes, the upstream analysis component can move on to its processing of other time steps.

Fig. 6 presents preliminary results of our transactional work providing resilience for management operations. A full discussion of this research appears elsewhere [14], but we feel it is important to highlight the costs associated with performing transactions in these environments. The *x*-axis represents the core ratio between writers and readers (e.g., 512 writers to 4 readers) and the *y*-axis represents the time taken to complete a transaction. Results show that

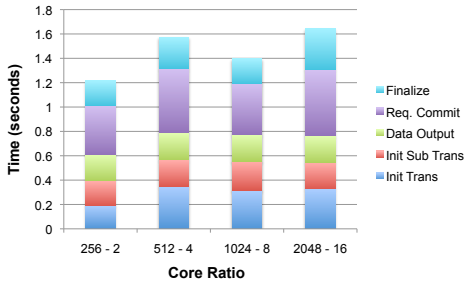

Figure 6: Microbenchmark of Resilience Protocol Overhead

Table II: Experiment Data Sizes

Node Count	Atoms	Data size
256	8,819,989	67 MB
512	17,639,979	134.6 MB
1024	35,279,958	269.2 MB

the solution provides good scalability and on-going work is making transactions cheaper without sacrificing resilience.

2) Using Containers to Improve Data Movement: We demonstrate the utility of containerized execution with experiments in which active management controls two metrics: container latency and end-to-end latency. We run weak-scaling experiments as the number of atoms LAMMPS simulates and outputs directly affects the time required by analysis routines. Table II shows the relation between node count and data size (data size output per timestep).

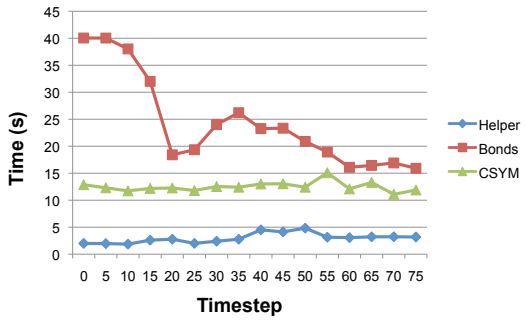

Figure 7: Events emitted for 256 simulation and 13 staging nodes

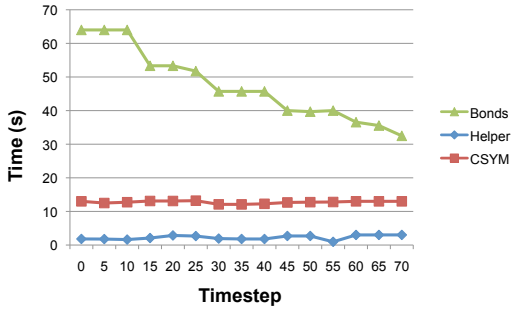

Figure 8: Events emitted for 512 simulation and 24 staging nodes

Container Latency. In Section III-E, we outline how container latency can be used to determine a bottleneck in the I/O pipeline. Active container management can control such latencies and thus avoid such bottlenecks. For this scenario,

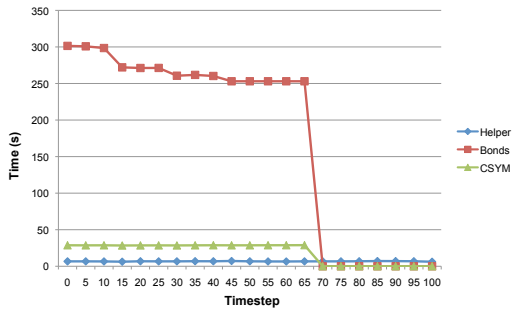

Figure 9: Events emitted for 1024 simulation and 24 staging nodes

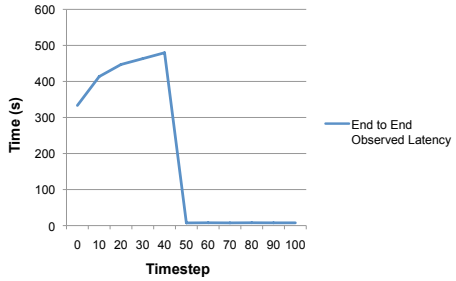

Figure 10: End-to-End Latency

LAMMPS output steps are generated more frequently than normal, every 15 seconds, to show capabilities even under stress. To avoid slowing down the pipeline, management must ensure that each container in the I/O pipeline can sustain this rate. Figure 7 shows the effects of increasing the Bonds container, the bottleneck in this instance. Since there are no spare resources to increase the Bonds container, the global manager first issues a decrease to the LAMMPS Helper container. Fortunately, LAMMPS Helper is fast enough to not experience a significant slowdown from this decrease. In other words, it is currently over-provisioned. Upon completion of these management actions, as expected, the latency for the Bonds container decreases. A transient issue for some runs, however, was delay due to DataTap pausing writers during a decrease. Fig. 7 shows a temporary increase in latency in the Bonds container after increasing its resources. This indicates the need for future research in less aggressive consistency methods.

Figures 8 and 9 show the Bonds container converging to the ideal rate. In fig. 8, there were insufficient resources but the simulation completed before any queue overflows occurred that would have blocked the pipeline. This was not the case with Fig. 9, however, and the containers runtime recognized the situation and moved the Bonds and Csym containers offline. Both of these experiments had 4 spare staging nodes at the start of the experiment.

End-to-End Latency. With this experiment, we evaluate end-to-end latency for the I/O pipeline by measuring the time it takes for each timestep to move through the pipeline. This includes data transfer times between the containers and compute times in containers. We use the same configuration as in the experiment depicted in Figure 9. In figure 10, despite increasing the bottleneck container, the end to end latency is increasing as data is still spending a large amount of time in the queue. Once the spare resources have been used and the Bonds container is moved offline, we see a sharp decrease in the end to end latency as the bottleneck is pruned from the data path.

V. Conclusions and Future Work

The I/O container abstraction and implementation developed in our work is important for several reasons. They make it possible to run dynamic online analytics and visualization tasks for simulations executing on high end machines. For such tasks, containers provide the functionality needed to run them at supervised levels of performance, without requiring tasks to be rewritten or reconfigured, and without requiring end users to laboriously 'size' the resources these tasks require. Using I/O containers also makes it easier to deploy and run entire scientific analysis pipelines, guided by end-to-end global properties like the delay science users experience between when simulation output occurs to when analyzed data is presented.

The current implementation of I/O containers uses a lightweight monitoring infrastructure to gather online performance information and hierarchical methods to make distributed management decisions. Its efficient management protocols are shown able to manage I/O pipelines effectively and in ways that improve analytics performance. Initial results with generalized protocols providing resilient management operations, using transactional constructs, are shown to give managers consistent views of current I/O pipeline state and resources.

Further issues driving our future work include the following: How to support stateful rather than stateless analytics methods; how to deal with more complex I/O pipelines; and how to place and co-locate containers on the petascale machine to reduce simulation-to-analytics data movement and taking into account node and interconnect topologies.

VI. Acknowledgements

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] 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*. IEEE Computer Society, 2003.

[2] W. X. Wang, Z. Lin, W. M. Tang, W. W. Lee, S. Ethier, J. L. V. Lewandowski, G. Rewoldt, T. S. Hahm, and J. Manickam, "Gyro-Kinetic simulation of global turbulent transport properties in tokamak experiments," *Physics of Plasmas*, vol. 13, no. 9, p. 092505, 2006.

[3] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng, "Datastager: scalable data staging services for petascale applications," *Cluster Computing 2010*, 2010.

[4] 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, p. 24.

[5] 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.

[6] C. Docan, M. Parashar, and S. Klasky, "Dataspaces: an interaction and coordination framework for coupled simulation workflows," in *HPDC*. ACM, 2010, pp. 25–36.

[7] M. Hereld, M. E. Papka, and V. Vishwanath, "Toward simulation-time data analysis and i/o acceleration on leadership-class systems," in *IEEE Symposium on Large-Scale Data Analysis and Visualization*, 2011.

[8] 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 *IEEE Symposium on Mass Storage Systems*, 2000, pp. 119–134.

[9] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. B. Ross, and N. F. Samatova, "Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data," in *Euro-Par (1)*, 2011, pp. 366–379.

[10] J. F. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf, "Managing variability in the io performance of petascale storage systems," in *SC*, 2010, pp. 1–12.

[11] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf, "Predata- preparatory data analytics on peta-scale machines," in *IPDPS*, 2009.

[12] F. Zheng, H. Abbasi, J. Cao, J. Dayal, K. Schwan, M. Wolf, S. Klasky, and N. Podhorszki, "In-situ i/o processing: A case for location flexibility," in *Parallel Data Storage Workshop (PDSW)*, 2011.

[13] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan, "Adaptable, metadata rich io methods for portable high performance io," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, may 2009, pp. 1 –10.

[14] J. Lofstead, J. Dayal, K. Schwan, and R. Oldfield, "D2t: Doubly distributed transactions for high performance and distributed computing," *Cluster Computing: To Appear*, 2012.

[15] M. Wolf, Z. Cai, W. Huang, and K. Schwan, "Smartpointers: personalized scientific data portals in your hand," in *SC*, 2002, pp. 1–16.

[16] G. Eisenhauer. Evpath. [Online]. Available: http://www.cc.gatech.edu/systems/projects/EVPath/

[17] K. Spafford, J. S. Meredith, J. S. Vetter, J. Chen, R. W. Grout, and R. Sankaran, "Accelerating s3d: A gpgpu case study," in *Euro-Par Workshops*, 2009, pp. 122–131.

[18] E. S. H. J. et. al, "CTH: A software family for multi-dimensional shock physics analysis," in *Proceedings of the 19'th International Symposium on Shock Physics*, R. Brun and L. Dumitrescu, Eds., vol. 1, Marseille, France, Jul. 1993, pp. 377–382. [Online]. Available: http://sherpa.sandia.gov/9231home/pdfpapers/issw.pdf

[19] J. R. Lange, K. T. Pedretti, T. Hudson, P. A. Dinda, Z. Cui, L. Xia, P. G. Bridges, A. Gocke, S. Jaconette, M. Levenhagen, and R. Brightwell, "Palacios and kitten: New high performance operating systems for scalable virtualized and native supercomputing," in *IPDPS*, 2010, pp. 1–12.

[20] H. Nasgaard, B. Gedik, M. Komor, and M. P. Mendell, "Ibm infosphere streams: event processing for a smarter planet," in *CASCON*, 2009, pp. 311–313.

[21] A. L. Varbanescu, A. M. Molnos, and R. van Nieuwpoort, Eds., *ISCA 2010 International Workshops, WIOSCA*.

[22] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: a platform for fine-grained resource sharing in the data center," in *NSDI 2011*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 22–22. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972488

[23] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP*, 2003, pp. 164–177.

[24] C. A. Waldspurger, "Memory resource management in vmware esx server," in *OSDI*, 2002.

[25] M. Wolf, H. Abbasi, B. Collins, D. Spain, and K. Schwan, "Service augmentation for high end interactive data services," in *CLUSTER*, 2005, pp. 1–11.

[26] W. Thies, M. Karczmarek, and S. P. Amarasinghe, "Streamit: A language for streaming applications," in *CC*, ser. Lecture Notes in Computer Science, R. N. Horspool, Ed., vol. 2304. Springer, 2002, pp. 179–196.

[27] D. Rosu and K. Schwan, "Faracost: An adaptation cost model aware of pending constraints," in *IEEE Real-Time Systems Symposium*, 1999, pp. 224–233.

[28] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: wait-free coordination for internet-scale systems," in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*.

[29] J. Borrill, L. Oliker, J. Shalf, and H. Shan, "Investigation of leading hpc i/o performance using a scientific-application derived benchmark," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, ser. SC '07. New York, NY, USA: ACM, 2007, pp. 10:1–10:12.

[30] G. Zheng, L. Shi, and L. V. Kalé, "Ftc-charm++: an in-memory checkpoint-based fault tolerant runtime for charm++ and mpi," in *CLUSTER*, 2004, pp. 93–103.

[31] R. Barik, Z. Budimlic, V. Cavé, S. Chatterjee, Y. Guo, D. M. Peixotto, R. Raman, J. Shirako, S. Tasirlar, Y. Yan, Y. Zhao, and V. Sarkar, "The habanero multicore software research project," in *OOPSLA Companion*, 2009.

[32] C. Wang, K. Schwan, V. Talwar, G. Eisenhauer, L. Hu, and M. Wolf, "A flexible architecture integrating monitoring and analytics for managing large-scale data centers," in *ICAC*, 2011, pp. 141–150.

[33] N. S. S. Interface, "https://software.sandia.gov/trac/nessie/."