Persistent Data Staging Services for Data Intensive In-situ Scientific Workflows

Conference Paper · June 2016

DOI: 10.1145/2912152.2912157

Melissa Romanus
Rutgers, The State University of New Jersey
14 PUBLICATIONS **64** CITATIONS
SEE PROFILE

C.S. Choong-Seock Chang
Princeton University
321 PUBLICATIONS **3,606** CITATIONS
SEE PROFILE

Ivan Rodero
Rutgers, The State University of New Jersey
99 PUBLICATIONS **1,009** CITATIONS
SEE PROFILE

Fan Zhang
Rutgers, The State University of New Jersey
19 PUBLICATIONS **262** CITATIONS
SEE PROFILE

Persistent Data Staging Services for Data Intensive In-situ Scientific Workflows

Melissa Romanus*
melissa@cac.rutgers.edu

Fan Zhang*
zhangfan@cac.rutgers.edu

Tong Jin*
tjin@cac.rutgers.edu

Qian Sun*
qiansun@cac.rutgers.edu

Hoang Bui*
H-Bui@wiu.edu

Manish Parashar*
parashar@rutgers.edu

Jong Choi†
choi@ornl.gov

Saloman Janhunen†
jjanhune@pppl.gov

Robert Hager‡
rhager@pppl.gov

Scott Klasky†
klasky@ornl.gov

Choong-Seock Chang‡
cschang@pppl.gov

Ivan Rodero*
irodero@rutgers.edu

*RDI^2, Rutgers University
Piscataway, NJ 08854

†Oak Ridge National Lab
Oak Ridge, TN 37831

‡Princeton Plasma Physics Lab
Princeton, NJ 08540

ABSTRACT

Scientific simulation workflows executing on very large scale computing systems are essential modalities for scientific investigation. The increasing scales and resolution of these simulations provide new opportunities for accurately modeling complex natural and engineered phenomena. However, the increasing complexity necessitates managing, transporting, and processing unprecedented amounts of data, and as a result, researchers are increasingly exploring data-staging and *in-situ* workflows to reduce data movement and data-related overheads. However, as these workflows become more dynamic in their structures and behaviors, data staging and in-situ solutions must evolve to support new requirements.

In this paper, we explore how the service-oriented concept can be applied to extreme-scale in-situ workflows. Specifically, we explore persistent data staging as a service and present the design and implementation of DataSpaces as a Service, a service-oriented data staging framework. We use a dynamically coupled fusion simulation workflow to illustrate the capabilities of this framework and evaluate its performance and scalability.

1. INTRODUCTION

High-performance computing (HPC) systems have revolutionized simulation-based science. Complex coupled simulation workflows running in-situ and at scale can provide dramatic insights into naturally occurring phenomena and engineering systems with higher resolution and finer-grained accuracy than ever before. However, the increased scale and complexity of such workflows present new challenges that impact their ability to run in a reasonable amount of time on present-day hardware. These challenges include the effective management of both the coordination of workflow components and the large amount of data that such workflows generate, process, and exchange.

For example, workflows can exhibit varying degrees of coupling and related coordination requirements, ranging from loose coupling, where the component simulations execute asynchronously with occasional data exchanges, to tighter coupling scenarios, wherein component simulations may exhibit lockstep execution behavior and frequent data exchange. As a result, there is a critical need for abstractions and efficient mechanisms that can effectively support these dynamic and varied coordination and data management requirements at scale.

Recent research has explored in-situ, in-memory techniques, based on data staging approaches, as a means of addressing the coordination and data management challenges for complex workflows [22, 27, 12]. For example, our previous work on DataSpaces [9] is an in-memory data staging implementation and has been used to support in-situ/in-transit workflows [26, 25]. Other staging implementations that have emerged include DataStager [2] and IOFSL [6]. However, the execution of these staging solutions is tightly coupled with the execution of the applications in the workflow. As a result, they cannot support workflows exhibiting complex dynamic behavior, such as the heterogenous launching of multiple instances of component applications over time and the data exchanges that must occur between them.

To address these requirements, we have extended the DataSpaces framework to provide scalable, persistent data staging as a service. Service-based staging allows application instances to connect to the service at the time they are launched and provides a way for previous, current, and future instances of applications to coordinate and share data. We believe that this persistent data staging service can provide a new level of flexibility to the formulation and execution of in-situ workflows and can enable new classes of coupled simulation workflows. In this paper, we present the

DIDC'16, June 01 2016, Kyoto, Japan
© 2016 ACM. ISBN 978-1-4503-4352-7/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2912152.2912157

XGC coupled fusion simulation workflow [5, 7] to motivate the need for a persistent staging service, and use it to drive the design, implementation, and evaluation of DataSpaces as a Service. DataSpaces as a Service provides the following attributes:

- **Dynamic**: Applications can dynamically connect to and disconnect from the staging service. We define new mechanisms to allow the coupled applications that are part of the workflow to join and leave the persistent data-staging service without impacting other component applications.
- **Persistence**: The staging service and the staged data remains persistent across instances of the component applications. Applications can join and leave the staging service whenever they need access to it.
- **Efficiency**: Write performance is optimized by routing data needed by a requesting application to the 'closest' staging servers. Access patterns can be learned and anticipated.
- **Resilient**: The staging service can be backed up and restarted as needed.

The rest of this paper is organized as follows. In Section 2, we discuss the XGC workflow and motivate the need for persistent staging as a service. In Section 3, we present the design and implementation of DataSpaces as a Service. In Section 4, we present an experimental evaluation and conceptual validation using the XGC workflow. In Section 5, we present related work in the areas of workflow management and data staging. In Section 6, we conclude the paper and outline future research directions.

2. COUPLED SIMULATION WORKFLOWS

As noted in Section 1, emerging coupled simulation workflows are exploring new and diverse coupling behaviors for enhanced scientific simulation, which results in challenging coordination and data management requirements. For example, the component applications of a workflow may be instantiated (and re-instantiated) multiple times during the execution lifetime of the workflow, but some may require the ability to coordinate and share data across each of these instantiations.

Existing workflows that exhibit static composition are forced to wait at times for simulations to reach a certain point before other components can begin or are prevented from performing any data processing until all relevant data has been collected. However, this is both time-intensive and data-size-dependent, since the end-to-end runtime is only as quick as its slowest component. As a result, scientists have begun to explore new possibilities that allow them to process data in-situ without impacting the overall workflow. One approach that has been gaining popularity in scientific communities is the idea of implementing different data querying, processing, and exploration services that operate on data as it is created, rather than waiting for a specific cue from a workflow stage or component.

In this section, we describe the XGC1-XGCa fusion simulation workflow. Although this workflow currently contains only two applications, it serves to illustrate the emerging coupling and coordination/data-management requirements noted above. We then use these requirements to motivate the persistent staging as a service solution, which is the focus of this paper.

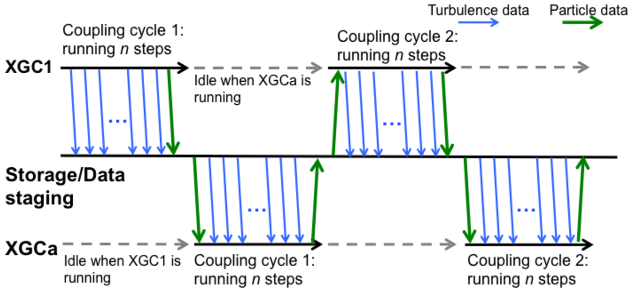
Turbulence data
Particle data
Coupling cycle 1: running *n* steps
XGC1
Idle when XGCa is running
Coupling cycle 2: running *n* steps
...
Storage/Data staging
XGCa
Idle when XGC1 is running
Coupling cycle 1: running *n* steps
Coupling cycle 2: running *n* steps

Figure 1: Data Flow Between Two Cycles of the XGC1-XGCa Coupled Workflow protect [11]

2.1 The XGC1-XGCa Fusion Workflow

The XGC1 [7] code is a well-known edge-fusion gyrokinetic simulation that employs a first principles kinetic particle-in-cell approach. The goal is to study the plasma edge physics for a magnetically confined tokamak plasma. The simulation is comprised of complex calculations that are both multi-physics and multi-scale, thus, running the simulation is only feasible through extreme-scale high-performance computing resources. XGCa [13] is asymmetric to the gyrokinetic turbulence code that comprises XGC1 and uses a 2-dimensional Poisson equation to calculate radial and poloidal electric fields that are self-consistent. The coupling of these two codes not only helps to correct errors and keep the plasma simulation from quickly diverging, but also allows for a seamless blending of fine-grained and coarse-grained models. The XGCa component can be considered a complimentary accelerator to XGC1, due to the fact that it uses much fewer particles and coarser-grained calculations.

The coupling between XGC1 and XGCa is very cyclic in nature. As illustrated in Figure 1, XGC1 runs for a number of time steps, then it stops, and XGCa runs using the turbulence and particle data generated by XGC1. After XGCa is finished, it writes new particle data which is consumed by XGC1 and the cycle repeats itself. As a result, XGC1 and XGCa each run once per coupling step. Scientifically, as XGC1 runs, the particles are allowed to move within the bounds of their physical properties (velocity, position, neighboring particles, etc). At each time step, the turbulence information is recorded. The number of time steps is dependent on when the plasma evolves from its initial transient phase to a quasi-steady turbulent state. At this point, all of the turbulence data has been written, and in addition, XGC1 writes the updated particle information and finishes its part of the cycle. XGCa then reads in all turbulence data, along with the state of the plasma (i.e., the particle data). Through a series of coarse-grained mesh calculations, XGCa uses the turbulence data to advance the plasma background enough to affect the turbulence and records the new particle positions at that point. XGCa writes the new particle data, and the coupling cycle ends. The updated particle data is read by XGC1 at the start of the next cycle.

A key feature of this workflow is that the data must be persistent even though the individual applications in the workflow are not. In addition, the workflow exhibits behaviors that make it ideal for in-situ processing. First, the data generated is quite large, as we will illustrate in Section 4 and would be inefficient if transferred to disk or communicated to all application ranks. Second, the data created by XGC1 is consumed immediately in the second half of the cycle, and the data created by XGCa is consumed immediately in the start of the next cycle. In this way, XGCa can be thought of as a data service for XGC1 and vis-versa.

2.2 Data Staging as a Service

State-of-the-art data staging techniques process data associated with workflow components in-situ using in-memory data staging [9, 6, 2]. In-memory staging uses the local-DRAM of compute nodes to store intermediate data objects and Remote Direct Memory Access (RDMA) for asynchronous data transfer [17]. Data staging has been shown to reduce the end-to-end time of the workflow and accelerate data discovery, primarily because it takes much longer to read and write files from the parallel file system (PFS) than it does to communicate between compute nodes over the high-speed network (HSN). Not all data generated at intermediate stages or time steps is necessary for the end result of a workflow. Thus, only critical data needed for offline processing or permanent storage needs to be written to the PFS, while intermediate data can be kept for varying durations in on-node memory.

Although existing data staging solutions have shown good performance in terms of scaling and overhead, they are still limited in the types of workflows they can support. The static nature of the current solutions means that new applications cannot be added to the workflow and others may sit idle on the machine until data becomes available. In reality, many scientific workflows would benefit from much more dynamic flexibility, i.e., applications may come online to compute an event of interest over multiple time steps, the results of a periodic uncertainty quantification calculation may steer future workflow calculations, new visualization techniques or analysis codes may be applicable to the data after a certain number of time steps, etc.

The ability to construct a means for workflow components to efficiently digest simulation data in-situ and allow for data feedback between workflow components has the potential to unlock new areas of scientific discovery. However, data staging must evolve to meet the increasingly dynamic demands of the latest extreme-scale scientific workflows. Thus, a new data staging infrastructure that facilitates the creation of more complex workflows and can support the addition or subtraction of component instances at the workflow's discretion is required. In the next section, we present an implementation of the staging as a service concept by adding new features to our existing data staging framework, DataSpaces.

3. IMPLEMENTATION

DataSpaces is a scalable data-sharing framework targeted at current large-scale systems and designed to support dynamic interaction and coordination patterns between large-scale scientific applications. Its framework builds an in-memory object storage repository by allocating memory from distributed compute nodes and create an abstraction of a virtual share-space. Applications can write to the staging area and read from the staging area using a simple put/get API. Moreover, DataSpaces utilizes advanced network capabilities (RDMA) and the underlining interconnect network to provide low latency and high throughput data transfer capability.

A series of compute nodes are responsible for executing the primary scientific simulation code, while a DataSpaces staging area is built on the dedicated compute nodes and utilizes any number of memory hierarchy schemes to effectively store data without going to the parallel file system. On-node DataSpaces servers can also run on a subset of cores of the nodes where the computations run, for faster read/write access. This work expands the capabilities of the existing DataSpaces software by creating a persistent staging area that supports varying numbers of component instances joining and leaving over the workflow lifetime. As an example, Figure 2 shows the the basic architecture of DataSpaces as a Service.

In this diagram, a scientific simulation runs from T0-T1. At T1, the scientific simulation is done and a visualization code starts. Since the computation is done, the visualization can run on the same cores and can utilize the Simulation data stored on-node or in the dedicated staging area without having to go to further levels of the storage hierarchy. The dashed lines represent the node interconnects (e.g., many recent high-performance infrastructures support RDMA networking), which will be exploited by the DataSpaces as a Service architecture automatically when available. In addition to improving read/write access, the DataSpaces as a Service model improves the overall utilization and efficiency of the underlying hardware, because reserved compute nodes are not idling while a different stage of the workflow runs. Additionally, network traffic and communication would be reduced by storing data generated by Stage 1 in the memory of the compute nodes on which Stage 2 will run.

3.1 As a Service Workflow

Figure 3 represents an example scientific workflow together with the DataSpaces as a Service model of persistence. In this model, applications can connect and disconnect to the staging area(s) only when they need to. Upward arrows indicate an application is getting data from the staging area, while downward arrows indicate the application is putting data into the staging area. Certain simulations or analytics components may do both. For example, an analysis code may run while one of the main simulations is running to correct for bias in the results. If this code detects a positive or negative bias, it may write a new parameter file to the DataSpaces as a Service staging area, which can be read by the simulation at some point in time and utilized to keep computation from diverging. As mentioned above, applications that do not overlap can utilize the same cores when applicable to increase overall resource utilization efficiency.

3.2 Dynamic Connection

In our previous implementation, all workflow components and their ranks (clients) had to make a connection to their designated master DataSpaces server before the workflow could begin execution. This implementation presents two potential bottlenecks. The first such bottleneck occurs in the scenario where there are thousands of clients attempting to connect to the master server. When the workflow begins, every client attempts to register with the master server simultaneously – for a large number of clients, the dramatic increase in communication and the massive number of requests can bring the registration process to a halt (i.e., the master cannot service these extreme-scale requests). The

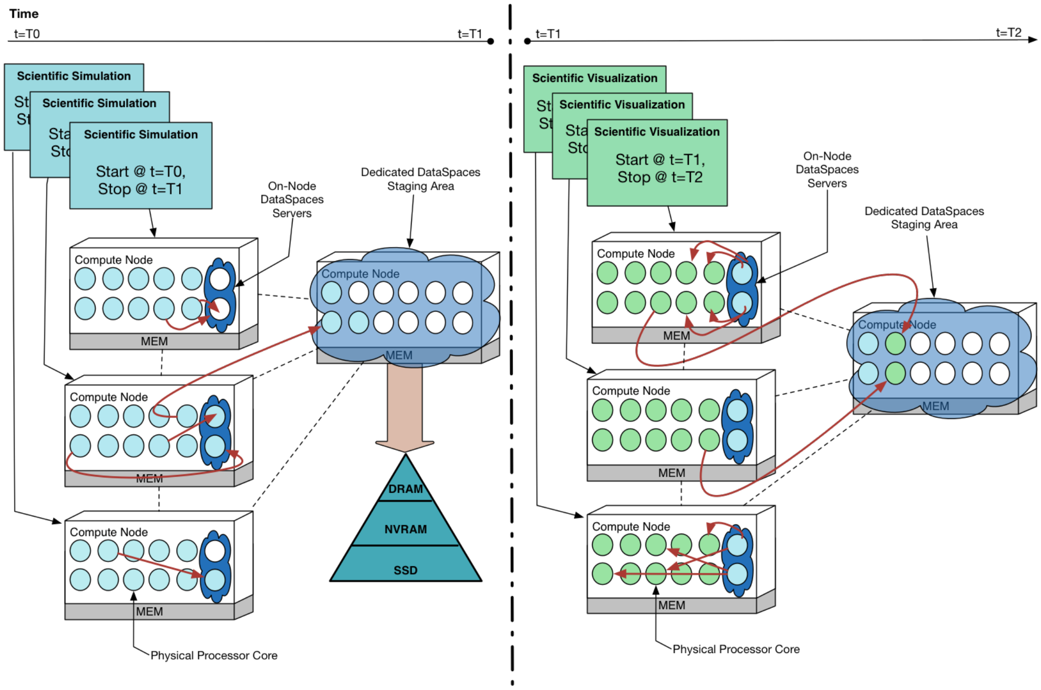

Figure 2: DataSpaces as a Service System Architecture. This diagram shows a lockstep coupling between a scientific simulation and a separate visualization application.

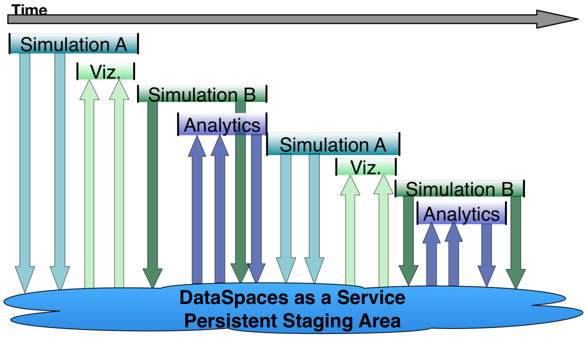

Figure 3: An example of a workflow made possible with DataSpaces as a Service. Applications run for fixed periods of time. Up arrows indicate data being read from the staging area by the application, while down arrows indicate data being written by the application to the staging area.

second bottleneck occurs when the master server attempts to distribute information to all of the connecting clients. For example, the master server is in charge of distributing the server list to all clients in a serial fashion. Although the size of the server list is small (a few KBs), too many messages originating from one server in a short period of time leads to a single point of slowdown. In order to circumvent these issues, we have made several changes to the registration process in the as a service model. In this model, clients (application instances) that flexibly attach and detach from the staging area are grouped according to the application to which they belong. When this grouping occurs, one of the clients is designated as a *Master Client*, which communicates with the Master Server on behalf of all of the clients in its grouping. Thus, only the Master Client will receive the list of staging servers, and it is responsible for distributing it amongst all of the instances of the specific application.

3.3 Data Write Optimization

In addition to supporting in-memory data staging on dedicated staging nodes, DataSpaces as a Service also implements the ability to co-locate in-memory data staging with application execution on the same set of compute nodes. The staging software utilizes a node-local storage resource, such as DRAM (in the current prototype), to cache and store application data that needs to be shared, exchanged, or accessed. This enables on-node data sharing for coupled simulation workflows that execute producer and consumer applications on the same compute nodes, which improves the locality of data access and reduces the amount of off-node data sharing/movement.

DataSpaces as a Service manages a number of byte-addressable shared memory segments as the storage space for data objects. The implementation uses the POSIX shared memory programming interface to create, open, map, and remove shared memory segments, which is portable and available on most Linux distributions. Applications running on the same node can utilize this on-node, in-memory data staging to share and exchange data. For example, a producer application writes data objects into the shared memory segment, which is write-once, read-only. Then, a consumer application running on the same node can connect to DataSpaces as a Service and acquire read access to the in-memory data objects and read the data from the shared memory segment.

The service-based DataSpaces implementation optimizes write performance by writing data to the “closest” staging server. After an application connects to DataSpaces, each of its processes becomes aware of the locations of the DataSpaces staging servers. On an extreme-scale computing system, the location information typically represents the network location of the compute node that executes the staging server. With the location information, each application process is able to compute the network distance between itself to all of the staging servers. When the process is writing data, it selects the closest staging server to write to. This location-aware optimization decreases the overall write time of each component process, allowing applications to continue progressing at a faster rate than without any optimization.

4. EXPERIMENTAL EVALUATION

To evaluate the performance and effectiveness of DataSpaces as a Service, we used the EPSI XGC1-XGCa workflow - a multi-physics plasma simulation workflow. In the following subsections, we review the data flow of this workflow and discuss the experimental setup and results achieved by integrating it with DataSpaces as a Service.

4.1 EPSI Workflow

As explained in Section 2 and illustrated in Figure 1, the XGC1-XGCa workflow executes for multiple coupling iterations. We briefly review the details of the workflow coupling here, with a focus on the computational behavior rather than its scientific significance.

In each coupling iteration, the workflow first executes XGC1 for n time steps (n is 20 in our experiments) to compute turbulence data and particle state and then executes XGCa for m time steps to evolve the state of plasma. The sharing of turbulence data is one-way from XGC1 to XGCa. XGC1 writes turbulence data at each time step of its execution, which is read by XGCa in the subsequent execution step. The sharing of particle data is two-way. Both XGC1 and XGCa write particle data at the end of their execution and need to read particle data generated by the other application at the beginning of their execution. In order to demonstrate the DataSpaces as a Service capabilities, we have implemented the workflow such that corresponding processes of both XGC1 and XGCa execute on the same set of compute nodes and write turbulence and particle data into a node-local memory buffer. Because of this characteristic, the EPSI workflow can take advantage of on-node memory sharing mode. Turbulence and particle data are distributed so there is no need for out-of-node data exchange.

Although most of the following experiments illustrate the potential performance gains of a service-based staging approach, it is important to note that they also demonstrate the ability for the two workflow applications (and their associated processes) to dynamically join and leave persistent on-node staging areas. In this approach, the ranks of XGC1 that are responsible for certain regions of domain computation are run on the same compute nodes as the ranks of XGCa that need to consume data for the same area of interest. Although XGC1 and XGCa completely terminate each time they are done, DataSpaces as a Service is utilized to make the DRAM persistent and sharable in this lockstep coupling workflow.

4.1.1 *Experimental Setup*

To evaluate the performance of our framework with the EPSI workflow, we deployed DataSpaces as a Service on the ORNL Titan Cray XK7 supercomputer. Titan consists of 18,688 compute nodes with a total of 710 Terabytes of memory. Each compute node has a 16-core AMD Opteron CPU and 32GB DRAM memory. Compute nodes are connected using Cray's high-performance Gemini network. We utilize a Lustre (PFS) working directory that has access to 1008 OSTs and 14PB of usable disk space. The default values for the stripe count (4) and stripe size (1MB) were not adjusted.

Table 1 summarizes the experimental setup. The number of application processes for XGC1/XGCa is varied from 1K up to 16K. The workflow runs for 2 coupling iterations, both XGC1 and XGCa execute for 20 time steps per coupling iteration. In each coupling iteration, the size of particle data read by XGC1 and XGCa ranges from 4GB to about 65GB, and the total size of turbulence data read by XGCa ranges from 12GB to about 202GB. To demonstrate the benefits of on-node data sharing, we compare the performance with both the *disk-based* approach, which exchanges data through disk files, and the *server-based* approach, which exchanges data through a set of dedicated staging compute nodes.

4.1.2 *Results and Discussion*

Table 2 and Figure 4 present the performance for exchanging *particle* data between the XGC1 and XGCa applications. The results show significant performance improvement for on-node data sharing when compared with the two other approaches. As shown by Tables 2, the *on-node* data sharing approach decreases the total time for writing particle data by 99% on average when compared with the *disk-based* approach and by 93% on average when compared with the *server-based* approach (i.e., DataSpaces without as a service modifications). As shown by Figure 4, on-node data sharing decreases the total time for reading particle data by 98% on average when compared with the *disk-based* approach and by 92% on average when compared with the *server-based* approach. This significant performance advantage is due to the in-memory local data caching, which does not require moving data off-node. In the *disk-based* approach, particle data needs to be moved and written to the storage system, while in the *server-based* approach, particle data needs to be transferred across the HSN to the staging servers.

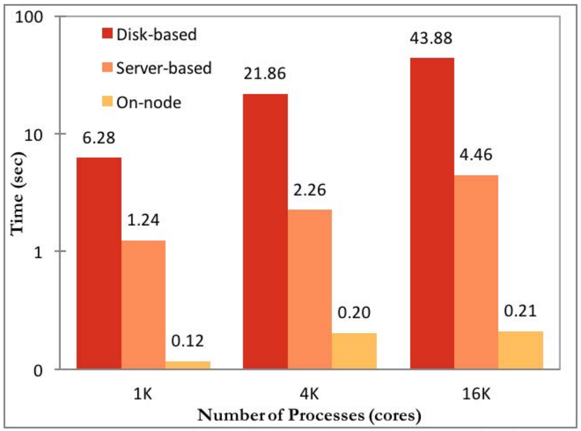

Figure 4: Total time taken reading particle data.

Table 3 and Figure 5 present the performance of exchanging *turbulence* data between XGC1 and XGCa. As shown by Table 3, the *on-node* approach decreases the total time for writing turbulence data by 99% on average when compared with a *disk-based* approach and by 31% on average when compared with the *server-based* approach. One observation is that when compared with the *server-based* approach, the performance improvement of writing turbulence data is not

	Setup 1	Setup 2	Setup 3
Num. of processor cores	1024	4096	16384
Num. of coupling iteration	2	2	2
XGC1 num. of steps (per iteration)	20	20	20
XGCa num. of steps (per iteration)	20	20	20
Size of particle data written/read by XGC1 (per iteration)	4.05 GB	16.21 GB	64.85 GB
Size of particle data written/read by XGCa (per iteration)	4.05 GB	16.21 GB	64.85 GB
Size of turbulence data write by XGC1 (per iteration)	0.19 GB	0.19 GB	0.19 GB
Size of turbulence data read by XGCa (per iteration)	12.63 GB	50.52 GB	202.09 GB

Table 1: Experimental setup for the evaluation of XGC1-XGCa coupled simulation workflow.

	Setup 1	Setup 2	Setup 3
Disk-based (seconds)	35.792	90.062	425.059
Server-based (seconds)	1.865	2.283	3.781
On-node (seconds)	0.097	0.131	0.316

Table 2: Total time taken writing particle data.

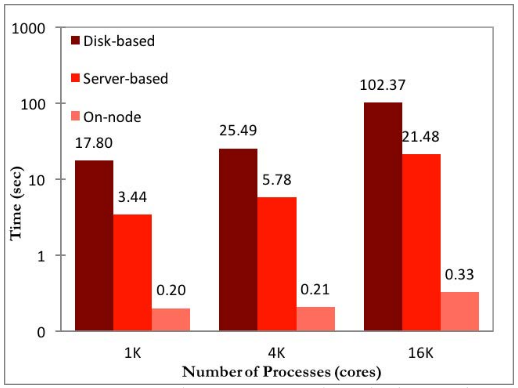

Figure 5: Total time to read turbulence data.

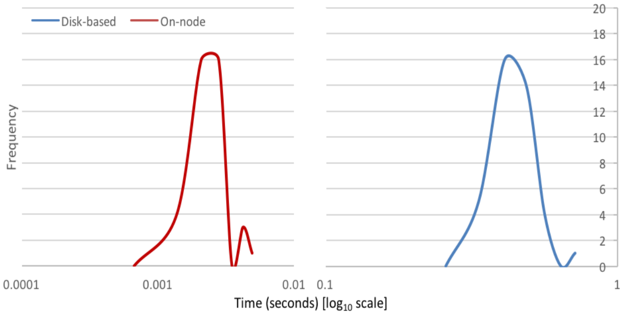

Figure 6: Stability of On-node vs Disk-based Approach for Writing Turbulence Data over 40 Time Steps on 16K Cores.

as significant as that of writing particle data. The reason is that the size of the turbulence data written by the XGC1 application is very small (as shown in Table 1). As a result, the time for transferring turbulence data to the staging servers is also small. Figure 5 presents the performance for reading turbulence data. On-node data sharing decreases the total time for reading turbulence data by 99% on average when compared with *disk-based* approach and by 96% on average when compared with a *server-based* approach.

In addition to improved write and read performance, the service-based staging approach also allows the workflow to progress at a much faster rate. In a disk-based approach, the component applications must continue to execute after computation is finished as a file for PFS is opened, written to (or read from), and ultimately closed. In an on-node approach, it is much faster to write and read data *objects* from DRAM. It eliminates time wasted in file open and close routines, and the faster write time allows XGC1 to shut down faster and XGCa to start up sooner than in a disk-based approach.

Figure 6 compares the *on-node* and *disk-based* approaches when writing turbulence data over a total of 40 time steps on 16K cores. The standard deviation of values about the mean for each data set is shown, where the time taken to write the data in seconds is displayed as a logarithmic scale. The frequency (horizontal access) indicates the number of time steps falling within each std. deviation. A log scale along the vertical axis was necessary to display the correlation between the two approaches, and the physical tick mark spacing was shortened between the 0.01 to 0.1 gap.

The standard deviation of the on-node write times over the 16k cores was calculated as 7.04e−0.4 and 7.63e−0.2 for the disk-based approach. Thus, the on-node approach experiences approximately two orders of magnitude less variability. The larger fluctuations in the disk-based approach illustrate the performance degradation that can occur on parallel file systems when many users are performing large numbers of I/O operations. This result shows how an underperforming PFS can severely impact the execution of workflow components and ultimately delay the end-to-end runtime of the workflow. At longer time scales, these variations propagate through the components and aggregate as time advances.

Writing to DRAM on a node that is exclusive to the application not only provides faster performance but also provides a more consistent approach. The small secondary hump of the on-node plot is due to 5 time steps falling at a higher range than the mean. The on-node approach does introduce some overhead to allow for in-situ operation, whereby processes must write the turbulence data for the *i-1* and *i+1* plane as well as the *i*-plane for use in the subsequent step. This can sometimes cause contention between processes on a node accessing slices of shared memory. How-

	Setup 1	Setup 2	Setup 3
Disk-based (seconds)	36.228	31.236	16.430
Server-based (seconds)	0.024	0.039	0.029
On-node (seconds)	0.016	0.029	0.019

Table 3: Total time of writing turbulence data.

ever, the plot shows that this variability is fairly uncommon and adds only 0.001 sec delay. Future work will investigate if there are opportunities for optimization.

5. RELATED WORK

There are a number of existing workflow managers targeting scientific applications, such as Makeflow [3], DAGMan[19] and Swift[21]. These managers emphasize the many-task computing type of workflow running on grid and cloud environments. These types of workflows use disk files to simplify data coupling patterns. Because the workflow relies on physical disks, the I/O performance often suffers. For applications that require tight data coupling, this approach proves to be particularly ineffective, because the latency of data access (reading/writing from a physical disk) prevents the workflow from finishing in a reasonable amount of time.

Recent research has been conducted to bridge the gap between memory and disk I/O performance. RAMDisk[24] has been used as cache in HPC systems in order to provide low latency data access to coupled applications. RAMDisk combines DRAM from compute nodes to form a memory pool and presents it to user applications as a traditional filesystem on physical hard drives. Applications write and read files to RAMDisk using regular POSIX APIs. However, one drawback to this approach is that RAMDisk cannot handle a large number of files due to its metadata writing operations. Thus, applications that exchange data using thousands of small files experience limited I/O performance with this method.

In order to accelerate the data-to-insight translation, a number of online data processing approaches have been proposed to perform in-memory data analysis, such as visualization [16], descriptive statistical analysis [20], and topology [4]. Data staging based frameworks, i.e DataStager [2], DataSpaces and ActiveSpaces [8], and JITStaging [1] cache application data in the DRAM of a staging area (a set of additional compute nodes) and perform data processing in-transit, which addresses the increasing performance gap between computation and I/O. These frameworks along with GLEAN [23] and Nessie [15] are notable for improving the parallel I/O performance by offloading output data from primary computing resource to staging area with support for asynchronous data transfers using modern networking techniques such as RDMA.

In-situ data analytics is also an emerging area of research. For example, GoldRush [28] utilizes idle CPU cycles available on simulation processor cores to perform in-situ data analysis. Damaris [10] implements middleware that supports in-situ parallel I/O and data visualization on dedicated cores. Burst Buffers [14, 18] create an intermediate layer of SSDs between the compute nodes and PFS, and Active Flash [22] utilizes a compute node's local SSD to support in-situ data analysis. Previous research on DataSpaces has investigated the performance of extending in-situ/in-transit data analysis with SSD-based data staging and deeper memory hierarchies [11].

6. CONCLUSION AND FUTURE WORK

In this paper, we describe the overall design and implementation of DataSpaces as a Service, a framework that provides in-memory data staging as a persistent service to scientific workflows. We demonstrate that using in-memory staging techniques as well as providing a persistent staging service can help coupled scientific workflows to scale up and to speed up the scientific discovery cycle. This method of in-situ processing allows multiple users and multiple applications to access the staging area, instead of the traditional in-situ method of compiling applications into one large binary so that they can share a communicator. We demonstrate improved scaling behavior up to 16K processors on ORNL Titan for the EPSI workflow. Our approach of using DataSpaces as a Service provides a very promising way of efficiently integrating standalone scientific simulations with scalable visualization and analysis codes.

For future research, we plan to expand DataSpaces as a Service to support deep memory hierarchies, as well as integrate data replication capabilities. Also, in the current implementation, data needed at the end of the workflow must be managed by the application. Our recent research efforts build upon the service-based approach to explore ways of exposing prefetching and post-job stage out operations to workflows, possibly by managing the movement of data throughout the various memory hierarchy layers.

7. ACKNOWLEDGEMENT

The research presented in this work is supported in part by National Science Foundation (NSF) via grant numbers CNS 1305375, ACI 1339036, ACI 1310283, ACI 1441376, ACI 1464317 and IIS 1546145, and by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the US Department of Energy Scientific Discovery through Advanced Computing (SciDAC) Institute for Scalable Data Management, Analysis and Visualization (SDAV) under award number DE-SC0007455, the DoE RSVP grant via subcontract number 4000126989 from UT Battelle, the Advanced Scientific Computing Research and Fusion Energy Sciences Partnership for Edge Physics Simulations (EPSI) under award number DE-FG02-06ER54857, the ExaCT Combustion Co-Design Center via subcontract number 4000110839 from UT Battelle, and via grant number DE-FOA-0001338, Storage Systems and Input/Output for Extreme Scale Science. The research at Rutgers was conducted as part of the Rutgers Discovery Informatics Institute (RDI^2).

8. REFERENCES

[1] H. Abbasi, G. Eisenhauer, M. Wolf, K. Schwan, and S. Klasky. Just In Time: Adding Value to The IO Pipelines of High Performance Applications with JITStaging. In *Proc. 20th International Symposium on High Performance Distributed Computing (HPDC'11)*.

[2] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. Datastager: scalable data

staging services for petascale applications. In *Proc. 18th International Symposium on High Performance Distributed Computing (HPDC'09)*, 2009.

[3] M. Albrecht, P. Donnelly, P. Bui, and D. Thain. Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, page 1. ACM, 2012.

[4] P.-T. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell. Interactive exploration and analysis of large-scale simulations using topology-based data segmentation. *Visualization and Computer Graphics, IEEE Transactions on*, 17, 2011.

[5] C. S. Chang and S. Ku. Spontaneous rotation sources in a quiescent tokamak edge plasma. *Physics of Plasmas*, 15(6), 2008.

[6] J. Cope, K. Iskra, D. Kimpe, and R. Ross. Bridging hpc and grid file i/o with iofsl. In *Applied Parallel and Scientific Computing*, pages 215–225. Springer, 2012.

[7] E. F. D'Azevedo, J. Lang, P. H. Worley, S. A. Ethier, S.-H. Ku, and C. Chang. Hybrid mpi/openmp/gpu parallelization of xgc1 fusion simulation code. In *Supercomputing Conference 2013*, 2013.

[8] C. Docan, M. Parashar, J. Cummings, and S. Klasky. Moving the Code to the Data - Dynamic Code Deployment Using ActiveSpaces. In *Proc. 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS'11)*, May 2011.

[9] C. Docan, M. Parashar, and S. Klasky. DataSpaces: An Interaction and Coordination Framework for Coupled Simulation Workflows. In *Proc. of 19th International Symposium on High Performance and Distributed Computing (HPDC'10)*, June 2010.

[10] M. Dorier, R. Sisneros, T. Peterka, G. Antoniu, and D. Semeraro. Damaris/viz: A nonintrusive, adaptable and user-friendly in situ visualization framework. In *Large-Scale Data Analysis and Visualization (LDAV), 2013 IEEE Symposium on*, Oct 2013.

[11] T. Jin, F. Zhang, Q. Sun, H. Bui, M. Romanus, N. Podhorszki, S. Klasky, H. Kolla, J. Chen, R. Hager, C.-S. Chang, and M. Parashar. Exploring data staging across deep memory hierarchies for coupled data intensive simulation workflows. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, May 2015.

[12] J. Kim, H. Abbasi, L. Chacon, C. Docan, S. Klasky, Q. Liu, N. Podhorszki, A. Shoshani, and K. Wu. Parallel in situ indexing for data-intensive computing. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 65 –72, oct. 2011.

[13] S. Ku, C. Chang, and P. Diamond. Full-f gyrokinetic particle simulation of centrally heated global itg turbulence from magnetic axis to edge pedestal top in a realistic tokamak geometry. *Nuclear Fusion*, 49(11):115021, 2009.

[14] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn. On the role of burst buffers in leadership-class storage systems. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*. IEEE, 2012.

[15] J. Lofstead, R. Oldfield, T. Kordenbrock, and C. Reiss. Extending scalability of collective io through nessie and staging. In *Proceedings of the sixth workshop on Parallel Data Storage*, pages 7–12. ACM, 2011.

[16] K.-L. Ma. In situ visualization at extreme scale: Challenges and opportunities. *Computer Graphics and Applications, IEEE*, 29(6):14–19, 2009.

[17] J. Pinkerton. The case for rdma. *RDMA Consortium, May*, 29:27, 2002.

[18] M. Romanus, R. B. Ross, and M. Parashar. Challenges and considerations for utilizing burst buffers in high-performance computing. *CoRR*, abs/1509.05492, 2015.

[19] D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. *Grid computing: Making the global infrastructure a reality*, 2003.

[20] D. Thompson and J. Bennett. Computing contingency statistics in parallel: Design trade-offs and limiting cases. 2010.

[21] S.-P. Tiberiu, B. Clifford, I. Foster, U. Hasson, M. Hategan, S. L. Small, M. Wilde, and Z. Yong. Accelerating medical research using the swift workflow system. *Studies in health technology and informatics*, 2007.

[22] D. Tiwari, S. Boboila, S. S. Vazhkudai, Y. Kim, X. Ma, P. Desnoyers, and Y. Solihin. Active flash: towards energy-efficient, in-situ data analytics on extreme-scale machines. In *FAST*, 2013.

[23] V. Vishwanath, M. Hereld, and M. Papka. Toward simulation-time data analysis and i/o acceleration on leadership-class systems. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on.*

[24] T. Wickberg and C. Carothers. The ramdisk storage accelerator: a method of accelerating i/o performance on hpc systems using ramdisks. In *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*. ACM, 2012.

[25] F. Zhang, C. Docan, M. Parashar, S. Klasky, N. Podhorszki, and H. Abbasi. Enabling in-situ execution of coupled scientific workflow on multi-core platform. In *Proc. 26th IEEE International Parallel and Distributed Processing Symposium*, 2012.

[26] F. Zhang, S. Lasluisa, T. Jin, I. Rodero, H. Bui, and M. Parashar. In-situ feature-based objects tracking for large-scale scientific simulations. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 736–740, Nov 2012.

[27] F. Zheng, J. Cao, J. Dayal, G. Eisenhauer, K. Schwan, M. Wolf, H. Abbasi, S. Klasky, and N. Podhorszki. High end scientific codes with computational i/o pipelines: improving their end-to-end performance. In *Proceedings of the 2nd international workshop on Petascal data analytics: challenges and opportunities*, PDAC '11, New York, NY, USA. ACM.

[28] F. Zheng, H. Yu, C. Hantas, M. Wolf, G. Eisenhauer, K. Schwan, H. Abbasi, and S. Klasky. Goldrush: resource efficient in situ scientific data analytics using fine-grained interference aware execution. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2013.