Adaptive Data Placement For Staging-Based Coupled Scientific Workflows

Qian Sun[†], Tong Jin[†], Melissa Romanus[†], Hoang Bui[†], Fan Zhang[†], Hongfeng Yu[‡], Hemanth Kolla[¶], Scott Klasky[§], Jacqueline Chen[¶], Manish Parashar[†]

[†]Rutgers Discovery Informatics Institute, Rutgers University, Piscataway, NJ, USA

[‡]Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588, USA

[¶]Sandia National Laboratories, Livermore, CA 94550, USA

[§]Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

ABSTRACT

Data staging and in-situ/in-transit data processing are emerging as attractive approaches for supporting extreme scale scientific workflows. These approaches improve end-to-end performance by enabling runtime data sharing between coupled simulations and data analytics components of the workflow. However, the complex and dynamic data exchange patterns exhibited by the workflows coupled with the varied data access behaviors make efficient data placement within the staging area challenging. In this paper, we present an adaptive data placement approach to address these challenges. Our approach adapts data placement based on application-specific dynamic data access patterns, and applies access pattern-driven and location-aware mechanisms to reduce data access costs and to support efficient data sharing between the multiple workflow components. We experimentally demonstrate the effectiveness of our approach on Titan Cray XK7 using a real combustion-analyses workflow. The evaluation results demonstrate that our approach can effectively improve data access performance and overall efficiency of coupled scientific workflows.

CCS Concepts

•**Information systems** → *Hierarchical storage management;* •**Computer systems organization** → *Real-time system specification;*

Keywords

Adaptive data placement, data access pattern, in-situ/in-transit, coupled scientific workflows, data staging

1. INTRODUCTION

Advanced coupled simulation workflows running at extreme scale on high end computing platforms are providing new capabilities and new opportunities for insights in a wide range of application domains. These workflows, which are composed of multiple coupled simulations, data analysis, visualization and other application components, are also presenting new challenges due to their scales, coupling and coordination behaviors and overall complexities. These challenges must be addressed before the potential of these workflows can be fully realized. For example, the S3D combustion simulation workflow [6] requires a variety of different, possibly concurrent, runtime analyses (e.g., descriptive statistics [21], iso-surface extraction [26], feature tracking [30], etc.) in order to derive insights from transient phenomena in the simulation data. These workflow components can exhibit distinct runtime data access behaviors and dynamic data exchange patterns. As a result, efficiently managing data placement, sharing, and exchange for such coupled simulation workflows has become a significant challenge.

Recent research efforts have used in-memory data staging and in-situ/in-transit data processing approaches to address these challenges. These approaches use a staging area that is composed of in-memory storage distributed across a set of staging cores/nodes on the system where the workflow is running, to enable runtime data processing, sharing, and exchange, as illustrated in Figure 1. The effectiveness of this solution however is sensitive to the data placement across the staging cores/nodes since data access latency can significantly impact the overall performance of the workflows – this is especially true for read-intensive workflows such as the S3D analysis workflow. Efficient data placement can be challenging when multiple components with different and possibly dynamic data access behaviors (in terms of data accessed, access frequency, and concurrency) are interacting and exchanging data at runtime, as is the case in the S3D analysis workflow. However, data placement in current data staging frameworks such as DataSpaces [9] and ActiveSpaces [10], is oblivious to the data access behaviors of the component applications that are part of the workflow. This can lead to inefficient data access and result in a significant performance impact due to load imbalance and increased contention.

In this paper, we propose an adaptive data placement approach to optimize the data access performance for staging-based scientific workflows. In this approach, we take advantage of application-specific data access patterns to adaptively place data with an awareness of the system network topology, so as to reduce data access costs and enable efficient data sharing, which in turn improves end-to-end per-

SC '15, November 15-20, 2015, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2807591.2807669

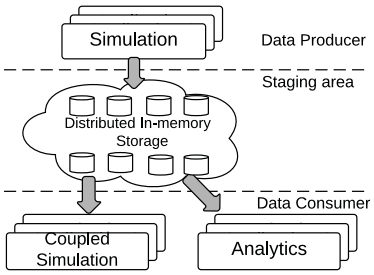


Figure 1: A simulation workflow composed of coupled simulations and analysis components, implemented using data staging.

formance. Specifically, we identify and characterize the dynamic data access patterns of data consumer applications at runtime using a combination of user provided hints and prior access behaviors. We also analyze the locations of compute nodes within the network topology of the system. Using insights from both of these analyses, we adaptively place data on the staging cores/nodes closest to the computational nodes that will access the data, while maintaining a balanced load across the staging cores/nodes. Furthermore, we dynamically replicate data to other staging nodes in order to resolve conflicting optimization requirements caused by concurrent accesses from multiple application components. This runtime data placement adaptation is performed *on-the-fly* while data is being transferred from the data producer application to the staging area.

We have developed a runtime system that implements this adaptive data placement approach on top of the DataSpaces [9] framework, and have deployed it on the Titan Cray XK7 system at Oak Ridge National Laboratory. We evaluate our runtime system using data access traces from real applications, and a production S3D combustion simulation workflow that couples S3D combustion simulation with two data analysis applications. We experimentally measure the data access performance, and demonstrate that our data placement approach effectively and efficiently adapts to dynamic data access patterns and significantly reduces the data access time as compared to other data placement approaches.

In this paper, we make the following contributions. (1) We present a data placement approach that leverages both application data access patterns and system network topology to dynamically and adaptively place data within a data staging area to reduce data access costs. (2) We implement and deploy a runtime system that realizes our adaptive data placement on Titan and demonstrate its effectiveness and performance using a production S3D combustion simulation workflow.

The rest of this paper is organized as follows. Section 2 presents a motivating application scenario. Section 3 describes our access pattern-driven and location-aware data placement approach. Section 4 presents the design and the implementation of our runtime system. Section 5 presents an experimental evaluation. Section 6 provides related work and Section 7 concludes the paper.

2. MOTIVATING APPLICATION WORKFLOW

The motivating application for this research is a scientific combustion simulation workflow; it is composed of one primary simulation, S3D [6], and many coupled analysis components. The S3D combustion simulation is a massively parallel code used to perform first principles direct numerical simulations of turbulent combustion. To glean fundamental insights from this simulation data, a variety of analyses are coupled with the simulation, such as iso-surface extraction [26], feature tracking [5], and volume rendering [29]. These analyses cover a broad set of algorithms that have heterogeneous data access patterns and requirements. Together, these analyses, plus the S3D simulation, comprise the overall workflow. We briefly describe the scientific background for two of the analyses components, in order to motivate the data access use cases that serve as drivers for this research.

Iso-surfaces Extraction: Extracting iso-surfaces of a varying scalar field in the computational domain is interesting and important for the S3D simulation since these iso-surfaces represent flame sheets in the turbulent flow. One challenge in extracting these iso-surfaces is that they are not volume filling, hence constructing them using a traditional marching cubes algorithm requires accessing the data from only a small portion of the entire data domain (less than 10%). However, the spatio-temporal fluctuations of turbulent flow cause the temporal iso-surface to experience "flapping." The result of this volatile behavior means that the required portion of the data domain needed for extraction may change accordingly over different time steps (i.e., it is not fixed). Furthermore, the domain scientist is often interested in the extraction of *multiple* iso-surfaces for different iso-values of the scalar field. In this case, multiple sub-regions of the data domain need to be accessed in the same time-step.

Feature Tracking: Direct numerical simulations resolve all the relevant spatio-temporal scales and provide information on the dynamics of many interesting features, such as auto-ignition kernels, expanding or contracting flames, and extinction regions. A feature can be typically identified and classified in a scalar field based on some critical points and a suitable threshold. Mostly, these features move and grow over time in the computational domain, but seldom extend to the full domain. Therefore, the feature tracking analysis can safely identify and track these features by accessing data of relevant sub-domains, under the guidance of corresponding thresholds. With the spatio-temporal changes of these features, the sizes and locations of these sub-domains change accordingly over time steps.

Both of these analyses exhibit dynamic data access patterns, which spatially range from small fractions of the sub-domain to the entire data domain, and temporally vary over different time steps. For these types of applications, existing data placement solutions that are oblivious to data access behaviors become inflexible and inefficient, especially in workflows with various analyses running concurrently. In the rest of this paper, we present an application-aware adaptive data placement approach that can take such complex and dynamic data access patterns into consideration to reduce data access costs and improve the overall time to solution for the large-scale simulation workflows.

3. ACCESS PATTERN-DRIVEN, LOCATION-AWARE DATA PLACEMENT

3.1 Overview of our approach

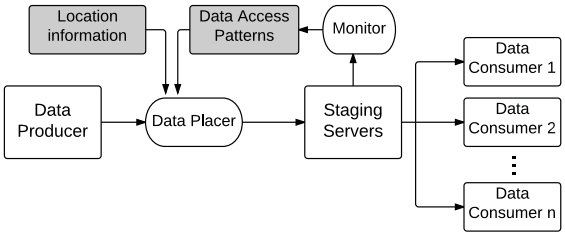


Figure 2: Overview of the adaptive data placement approach.

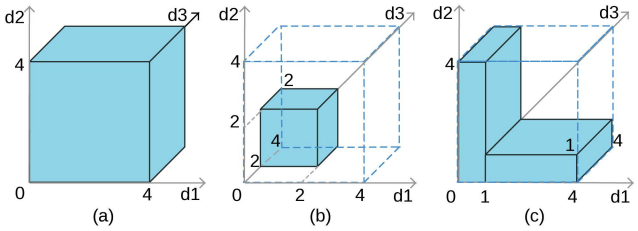


Figure 3: Examples of data access patterns in a 3D data domain. The shadowed area represents the data accessed by the data consumer applications.

The goal of our data placement approach is to optimize data access performance for staging-based in-situ/in-transit simulation workflows. To achieve this goal, we adapt the placement of data in the staging area to meet dynamic application requirements. We do this by analyzing data access behaviors at runtime and using this analysis to anticipate accesses. Based on this prediction, we place data close to the application components that will access. At the same time, we attempt to evenly distribute the data across the staging nodes to minimize data access contention. More specifically, we monitor the data access patterns of the consumer applications at runtime and use this information, along with information about the locations of the applications and staging nodes within the system network topology, to adaptively optimize data placement *on-the-fly* while the data is being transferred from the data producer application to the staging servers. Note that while adaptive data placement may increase the write latency, it will not impact the progress of the producer application, e.g., the simulation that is writing data. This is because data is written using an RDMA-based asynchronous write operation, which allows the application to continue its execution while the write is happening, thus effectively hiding the write latency from the application.

An overview of our adaptive data placement approach is presented in Figure 2. Our approach relies on two pieces of information, runtime data access patterns and location information. In the following subsections, we describe mechanisms for acquiring all this information with sufficient accuracy while ensuring that the associated overheads have minimal impact on the performance of the workflow.

3.2 Determining data access patterns

Data in most scientific simulations is defined on a discretized simulation domain and is represented as a multidimensional array. The data access patterns associated with such scientific applications have been studied in previous research [18]. We leverage this work to identify the data access patterns within simulation workflows. Specifically, we characterize the spatial data access patterns in an N-dimensional domain as one or multiple N-dimensional bounding boxes, where a bounding box is defined as follows:

$$S = \{(Coord_{start}, Coord_{end})_{d_1}, ..., (Coord_{start}, Coord_{end})_{d_n}\}$$

Here, the two coordinates – $Coord_{start}$ and $Coord_{end}$ – represent the boundaries of the bounding box in each dimension d_i. Such a bounding box representation is used in many scientific applications including combustion (S3D), fusion (GTC, GTS, XGC-1) and Magnetohydrodynamics (MHD) (Pixie3D) simulations, and data access patterns in these applications can be expressed as combinations of one or more bounding boxes. We intend to extend this approach to other data representations, such as graphs.

For example, Figure 3 shows the data access regions (shadowed in the figure) of two application processes p_1 and p_2, denoted as S_{p_1} and S_{p_2}, respectively. In this Figure, the combined data access region can be the entire data domain (Figure 3(a)) that is described as $S = \{(0,4)_{d1}, (0,4)_{d2}, (0,4)_{d3}\}$, a regular sub-domain (Figure 3(b)) that is described as $S = \{(0,2)_{d1}, (0,2)_{d2}, (2,4)_{d3}\}$, or an irregular sub-domain (Figure 3(c)) that is described as $S = S_{p_1} \bigcup S_{p_2}$, $S_{p_1} = \{(0,1)_{d1}, (0,4)_{d2}, (0,4)_{d3}\}$ and $S_{p_2} = \{(1,4)_{d1}, (0,1)_{d2}, (0,4)_{d3}\}$.

Since data access patterns might vary over time as the simulation evolves, we have to efficiently identify these patterns at runtime. To do this, we leverage the observation that data accesses in simulations typically follow regular patterns and evolve predictably over time. We monitor data accesses at the staging servers and compare them across consecutive time steps to anticipate subsequent accesses. Specifically, we track the changes in the boundary of the accessed region, denoted as K_{d_i}, occurring along each dimension d_i of a N-dimensional bounding box in time steps $T-1$ and $T-2$. We then predict the boundary of the accessed region in time step T to satisfy:

$$Coord_{d_i}^{T} = Coord_{d_i}^{T-1} + K_{d_i}, K_{d_i} = Coord_{d_i}^{T-1} - Coord_{d_i}^{T-2}$$

For example, in Figure 4, the bounding box accessed in time steps T_3 and T_4 are $\{(5,8)_{d1}, (3,5)_{d2}\}$ and $\{(3,6)_{d1}, (2,6)_{d2}\}$, respectively. Using the approach described above, the predicted bounding box accessed in T_5 is $\{(1,4)_{d1}, (1,7)_{d2}\}$. This approach is simple yet effective and introduces low overheads, as our evaluation presented in Section 5 demonstrates.

We also allow users to provide *hints* about *when* and *how frequently* data access patterns change, based on their knowledge of the application. For an iterative application, *hints* for the main loop can be specified as follows:

$$H = \{TS_{start}, TS_{end}, frequency\}$$

Here TS_{start} and TS_{end} define the *start* and *end* time steps of a specific interval in the application's execution; $frequency$ defines how often the access pattern may change within that interval. For example, $H = \{1, 50, 10\}$ states that the application changes its data access patterns every ten time steps in the first 50 time steps. As a result, our runtime will capture the data access patterns at time steps 1, 11, 21, etc., and adapt data placement if required.

Hints are an optional enhancement that guide the placement when provided. However, they might be incorrect due to user mistake. For instance, if a user claims access pattern changes at a certain time step, but in reality, it does not. To minimize the impact of incorrect hints, we compare

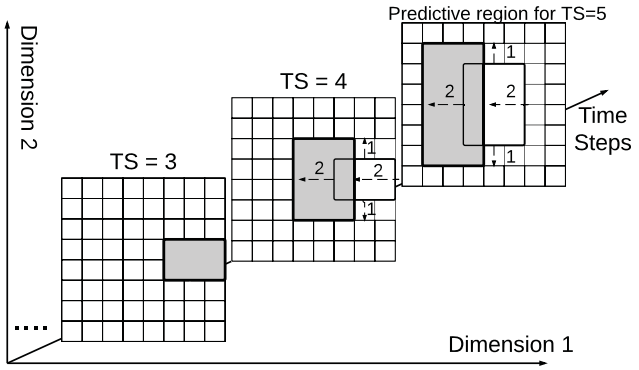


Figure 4: An illustration of our approach for anticipating data access patterns for a 2D data domain over time. The entire 8×8 data domain is written into the staging area by the producer application, and the gray regions indicate data read by a data consumer application. Our algorithm tracks changes in the accessed region along both dimensions of the data access region during the 3rd and 4th time steps, and anticipates the region that will be accessed during the 5th time step.

the data access patterns in the current and previous time steps. If they are the same, no changes on placement; thus, no effect on efficiency. In the case when a user is unaware of the pattern change, our approach might fail to capture the pattern changes immediately. Efficient data placement will not be achieved until the new pattern is captured.

3.3 Acquiring location information

Data transmission costs are a significant part of data access costs. For large scale systems with a torus/mesh or fat-tree network topology, the chance of network contention/ congestion increases as the data transfer distance increases [3, 16], resulting in increased data transmission costs. Furthermore, in production HPC systems such as Titan, the system scheduler often allocates non-contiguous compute nodes to jobs in order to improve the overall resource utilization. As a result, compute nodes allocated to the same job might be located physically 'far away' from one another, resulting in high data transmission costs between processes executing on these nodes. Therefore, keeping track of the physical locations of allocated compute nodes and the distances between them, is important for achieving efficient data placement with low data transmission costs.

3.3.1 Identifying physical locations of nodes

The physical location of a node describes its relative position in the system network. We illustrate our approach for identifying a node's physical location using a 3D mesh/torus network topology as an example, because this topology is widely used in current large scale HPC systems. The physical locations of compute nodes in this network topology can be represented using *coordinates* – (X, Y, Z), and the distance between a pair of nodes can be measured as the shortest path along network links in all dimensions based on their *coordinates*. For example, in Figure 5, application process p_1 running on compute node $(0, 1, 0)$ can communicate with staging server S_2 running on compute node $(1, 1, 0)$ via a one-hop path along the x dimension; the distance between p_1 and S_2 is one. Similarly, the distance between p_1 and S_1 is four. In order to reduce data access costs, the data required

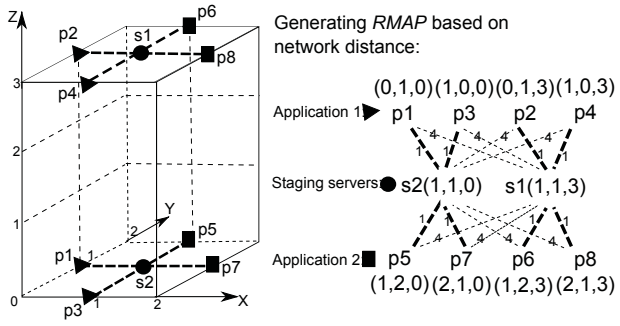


Figure 5: An example of the relationship map (RMAP) between two staging servers and two applications. Based on their locations in the network, application processes p_2, p_4, p_6 and p_8 are mapped to staging server S_1, while p_1, p_3, p_5 and p_7 are mapped to staging server S_2.

by p_1 should be placed as close as possible. Therefore, in this case, S_2 is a better choice than S_1.

3.3.2 Generating the relationship map – RMAP

The relationship map (RMAP) describes the proximity between data consumers and staging servers based on their physical locations within the system network topology. In RMAP, we map each data consumer process to a staging server that is physically close to it within the network. Meanwhile, we attempt to achieve load balancing across the servers by allowing each server to be mapped to approximately the same number of consumer processes. The mapping assumes that for the same consumer application, each consumer process accesses approximately the same amount of data, which is the case for the targeted applications. For example, as shown in Figure 5, eight processes running two applications access data from two staging servers. Based on their locations, application processes p_2, p_4, p_6 and p_8 are mapped to staging server S_1 with a distance of one; the other four application processes are mapped to server S_2.

Generating the relationship map described above can be reduced to the NP-complete Generalized Assignment Problem [22]. To minimize the impact on the simulation, we use a simple but effective greedy heuristic algorithm. Specifically, for each application process, our algorithm sorts the distances from that process to all the staging servers and then selects the 'closest' server to map to. We also set a constraint on the maximum number of processes that can be mapped to a single staging server to ensure load balancing. The complexity of this algorithm is $O(n{\cdot}mlog(m))$, where m, n represents the number of cores running the staging servers and the data consumer applications, respectively. Note that since the nodes allocated to an application do not change at runtime in current HPC systems, the RMAP only needs to be generated once when the workflow begins execution.

3.4 Determining data placement

Based on both pieces of information, the dynamic data access patterns and the static RMAP, our data placement approach places data close to the computation. Specifically, data that will be accessed by an application process is placed in advance on the staging server that the process is mapped to in the RMAP. For example, in Figure 6, based on the RMAP (derived in Figure 5) and the captured data access patterns for the two applications, p_4 is mapped to S_1 and accesses data d_4. As a result, our approach places d_4 on S_1

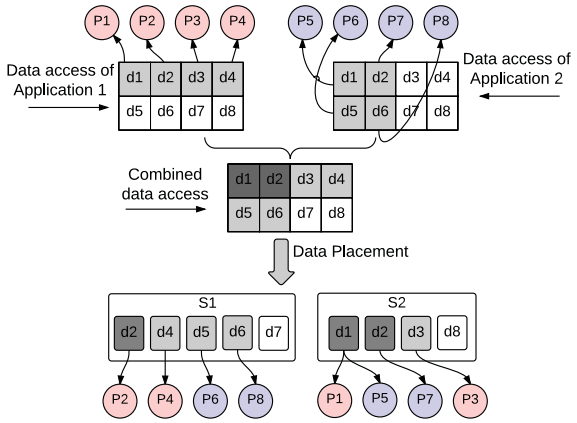


Figure 6: An example of data placement for two applications that read different data regions. The light gray regions in the full data domain are accessed by one application, while the dark gray regions are accessed by both applications.

to satisfy the requirements of p_4, as shown in the figure. The remaining data, i.e., data that is not identified by the access prediction mechanism as being accessed by any application, such as d_7 and d_8 in the figure, is evenly distributed across the staging servers to balance load. Note that at the beginning of execution (i.e., the first time step), there is no information available about data access patterns. As a result, we use a pre-configured data placement, such as row/column-based or chunking. In subsequent time steps, we can use the access behaviors captured in the previous time step to guide data placement. Once sufficient historical data about access behaviors is collected, the adaptive data placement approach can be applied based on the predicted data access patterns.

When multiple application components within a workflow concurrently access the same data, it might lead to conflicting data placement requirements. For example, in Figure 6, d_2 is accessed by two processes p_2 and p_7, which are mapped to S_1 and S_2, respectively. Placing d_2 on either S_1 or S_2 can only satisfy the requirement of one application process, resulting in higher data access costs for the other process. To address this problem, our approach dynamically replicates data and distributes replications to staging servers on demand. For the above example, a replica for d_2 is created and these two copies are distributed to both S_1 and S_2. Note that data that is replicated is read-only, thus data consistency is not an issue. In our approach, the number of replicas is determined by the data access requirement for each time step and thus adapts to the dynamic data access patterns. In addition, distinct data objects may have different numbers of replicas, which results in an efficient usage of storage space. For example, no extra replica is created for d_1 even though it is accessed by two processes p_1 and p_5, because both processes are mapped to S_2 and can share the same copy without conflict.

In practice, we may not be able to create as many replicas as required due to limited memory within the staging area. In this case, the incoming data will replace the data generated in previous time steps, especially those that have multiple replicas. The other option is to extend the capacity of staging area using NVRAM, SSD, or other available resources. Such architectures with deep memory hierarchy have been deployed on some HPC systems, such as Sith at

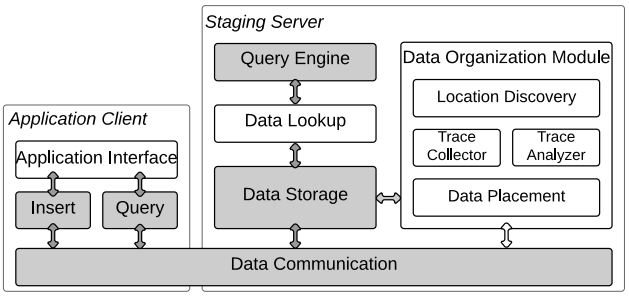


Figure 7: Architecture of the runtime system for adaptive data placement. The shadowed area represents components of DataSpaces that are reused by the runtime.

ORNL. We have explored data staging across deep memory hierarchy in other research presented in [14, 15].

4. IMPLEMENTATION OF A RUNTIME SYSTEM FOR ADAPTIVE DATA PLACEMENT

We have implemented a runtime system that incorporates our adaptive data placement approach into the DataSpaces data staging framework [9]. The schematic overview of the overall architecture of the runtime is presented in Figure 7. It leverages the *Data Communication Layer*, *Data Operation Layer* and *Data Storage* from DataSpaces, reusing its data transport, data insertion and querying, and data storage capabilities. Following the DataSpaces architecture, our system consists of a client-side subsystem that is co-located with the applications in a given workflow, and a server-side subsystem that runs on the staging cores. In this section, we describe the implementation details of the *Data Organization Module*, including *Location Discovery*, *Trace Collector*, *Trace Analyzer*, and *Data Placement*. We also detail the extended *Data Lookup* service that supports our data placement approach to efficiently serve data access requests.

4.1 Location Discovery

The *Location Discovery* component of the runtime is responsible for identifying the physical locations of nodes allocated to a workflow, including the DataSpaces staging servers. For example, in the Cray XK7 supercomputer, the location of a node can be represented by the *coordinates* of the attached Gemini router and be acquired by appropriate system calls (e.g., *rca_get_meshcoord*). A server that is selected as the master server, is responsible for gathering this information, generating a relationship map - *RMAP*, and distributing the *RMAP* to all staging servers. Based on the *RMAP*, each staging server sends messages containing its own information to the application processes that it is mapped to. It allows application processes to directly communicate with staging servers that are physically close to them. Since this location information does not change, the discovery process only needs to be performed once and in parallel with the execution of data producer application. Therefore, it has little impact on the overall workflow performance.

4.2 Trace Collector and Trace Analyzer

The *Trace Collector* and *Trace Analyzer* components are responsible for capturing dynamic data access patterns during runtime. Specifically, the *Trace Collector* logs every time an application process requests access to data. It extracts

the bounding boxes that describe the requested data regions as data access behaviors. The *Trace Analyzer* then analyzes such data access behaviors by comparing them with those in the previous time step in order to detect changes in the data access patterns. Data placement computations can be avoided if the data access patterns remain the same.

When *hints* are provided by the user, both the *Trace Collector* and *Trace Analyzer* are invoked only at selected time steps, at the beginning of a new execution phase that may introduce new data access patterns, thereby reducing some system overheads. In the absence of *hints*, the *Trace Collector* and *Trace Analyzer* may be invoked at every time step to collect the data access behavior and predict the future data access using the method described in Section 3.

4.3 Data Placement

The *Data Placement* component is responsible for making the data placement decision based on the information provided by the *Location Discovery*, *Trace Collector* and *Trace Analyzer* components. Specifically, the master server collects the data requirements for each staging server based on access patterns acquired by the *Trace Analyzer* component and determines an assignment of data objects to staging servers. This data assignment is then stored in a distributed hash table (DHT) constructed across all the staging servers. Indeed, the DHT itself is part of the original DataSpaces framework. We extend it in this work to support multiple data replicas, by allowing it to keep track of the locations of data objects and their replicas. The DHT is used by *Data Lookup* for runtime data placement and data query operations.

Specifically, the index for the DHT is derived from the applications' representation of the data domain. For example, in the case of a Cartesian mesh, a space-filling curve (SFC) is used to linearize the n-dimensional Cartesian coordinates into a 1-dimensional index space, which is then assigned to distinct staging servers. This information is used to distribute the generated data assignment to staging servers. Consequently, each staging server is assigned distinct segments of the index space, and maintains the DHT entries for the locations of data corresponding to that index space.

4.4 Data Lookup

The *Data Lookup* component builds on the existing Data Lookup Layer of DataSpaces. In this work, it has been extended specifically to support adaptively placing data based on the DHT, and to support accessing data replication.

The *Data Lookup* component hashes the spatial information associated with a data insert request (using the SFC) and determines the staging servers in the staging area to which the corresponding segments of the index space are assigned. It then uses the DHT to determine a list of locations where the inserted data should be stored, and routes the data to those appropriate staging servers. Similarly, for data retrieval queries, the staging servers that have the segments of the index space corresponding to the spatial information associated with the query, search their DHT entries for data locations. For a data object with multiple replicas, the most appropriate one that is stored close to the application process and does not violate the load balance across the servers, is selected and returned to application.

5. EXPERIMENTAL EVALUATION

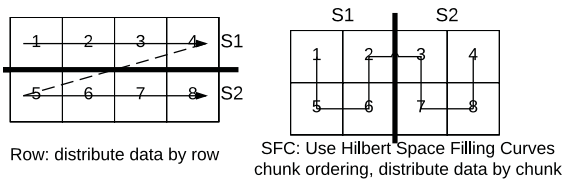


Figure 8: Data placement methods used in the experiments.

In this section, we present an experimental evaluation of our adaptive data placement approach, which is termed Pd-Loc (for **P**attern-**D**riven, **Loc**ation-Aware). We evaluated it using (1) data access traces from existing real applications and (2) the S3D combustion simulation-analysis workflow. Our experiments were performed on the Titan Cray XK7 supercomputer at ORNL. Titan has 18,688 compute nodes connected through a Gemini 3D torus interconnect; and each node has a single 16-core AMD 6200 series Opteron processor and 32GB of memory.

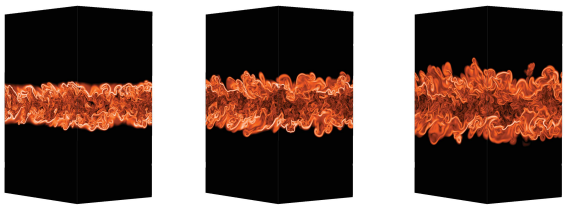
5.1 Evaluation using data access traces

This set of experiments evaluates the performance and effectiveness of our adaptive data placement (*Pd-Loc*) using data access traces from real application workflows that exhibit typical dynamic data access patterns. These data access traces were extracted from workflows where different analysis/visualization operations are performed on turbulent combustion datasets generated by S3D, a massively parallel combustion simulation code developed at Sandia National Laboratories.

We used traces from two use cases in the experiments: (1) extracting an iso-surface from the dataset that represents a 3D flame in stratified dimethyl-ether(DME)/air turbulent mixtures (as shown in Figure 9), and (2) performing volume rendering on the dataset that represents a 3D temporally-evolving planar slot jet flame with dimethyl-ether (DME) as the fuel (as shown in Figure 10).

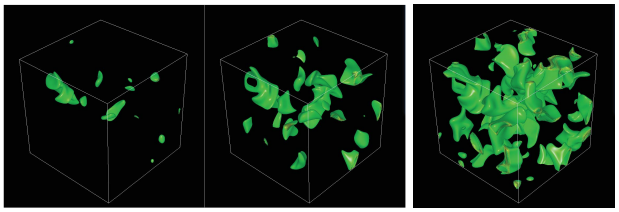
In these experiments, we coupled one data producer with one data consumer using two test application codes. One was responsible for writing data *to* the staging area (as a data producer), and the other for reading data *from* the staging area (as a data consumer), both following the data access patterns captured in the traces. The configuration of the experiments was as follows: the number of cores allocated for the producer and consumer were 8k and 4k respectively, and 256 cores were used for the staging servers. For simplicity, we used a normalized 3-dimensional data domain of size $2048 \times 2048 \times 2048$, with 16 different variables. A total of 256GB of data was generated for the entire domain and transferred to the staging area during a write.

As stated in previous sections, the efficiency of data reads impacts the overall performance of the workflow. Therefore, we measured the data read time at the data consumer application and evaluated the efficacy of our approach using two experiments. In these experiments, we compared the performance of our adaptive data placement method (*Pd-Loc*) with commonly used approaches for organizing multidimensional data generated by scientific applications across data staging nodes or parallel storage. Specifically, two data organizations[23] are popularly used: logically contiguous, e.g., *Row*, and chunking, e.g., Space Filling Curve (*SFC*), as illustrated in Figure 8. Detailed experiment results and comparisons are presented below. Please note that these results



(a) TS = 10. **(b)** TS = 50. **(c)** TS = 90.

Figure 9: A visualization of the evolving data access regions in the combustion dataset that represents a 3D flame in stratified dimethyl-ether (DME)/air turbulent mixtures. The data access regions that include the iso-surface (shown in red) are growing over time.



(a) TS = 1. **(b)** TS = 5. **(c)** TS = 9.

Figure 10: A visualization of the evolving data access regions in the combustion dataset that represents a 3D temporally-evolving planar slot jet flame with dimethyl-ether (DME) as the fuel. The regions of interest (shown in green) are growing and moving over time.

are averages over 20 test runs.

5.1.1 *Evaluation using the trace from the Combustion – Iso-surface extraction workflow*

In this experiment, we emulated the dynamic data access patterns extracted from the workflow composed of the combustion simulation and the iso-surface extraction component. In this scenario, scientists are interested in the regions corresponding to 50% of the total heat release. These regions would be useful for computing statistics conditioned on burning regions. It is possible to compute temperature, species mass fractions scalar dissipation rate, progress variable means, and pdfs conditioned on whether a region is burning or not.

Figure 9 shows a visualization of the data access regions that include the iso-surface structure at three selected time steps. The flame spreads over the entire X and Z axes and slowly grows along Y axis in both directions. Correspondingly, the data access regions that include the flame dynamically grow from 30% to 50% of the entire data domain, and the accessed data size increased from 76GB to 128GB. In this case, the data access traces include 100 time steps and the data access patterns changed every 10 time steps. The frequency of change in data access patterns was provided to the system in the form of user *hints*.

From Figure 11, we can observe that the performance of our adaptive approach (*Pd-Loc*) is relatively stable except for two time steps. One occurs at the 1st time step. This is because there is no prior knowledge about the data access patterns of the data consumer, and so we simply use *Row* placement initially. The other occurs at the 11th time step when the data access pattern changes, making the data placement based on the pattern from the previous time step ineffective. Once the data access pattern has been captured, our approach improves data access performance for the subsequent time steps.

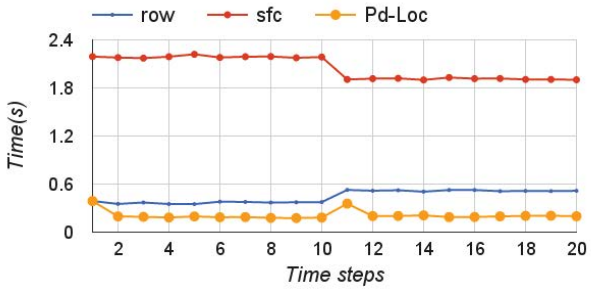


Figure 11: A comparison of data access times during the first 20 time steps using different data placement approaches.

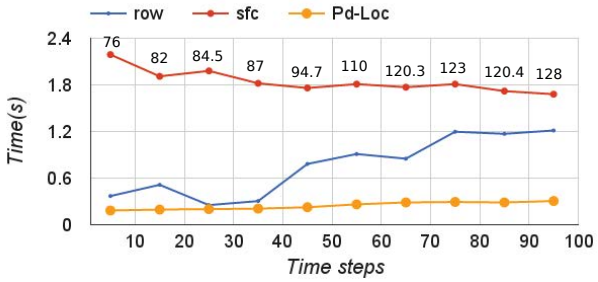


Figure 12: A comparison of the ***average data access time for every ten time steps*** that have the same data access patterns using different data placement approaches. The sizes (GB) of the accessed data regions for every ten time steps are also shown in the plot.

We performed this experiment over 100 time steps to better understand the data access performance with respect to the dynamic data access patterns over a longer time scale. When the data access patterns remain the same, the data access time for a specific data placement approach is almost constant. As a consequence, we simply compared the average data access time for every ten time steps (the interval during which all time steps have the same data access patterns), as plotted in Figure 12.

From this experiment, we can observe that *SFC* performs the worst. This is because it always places contiguous dataset on a few staging servers to preserve data locality. However, this approach can create hotspots, resulting in inefficient data access. In comparison with *Row* and *SFC*, *Pd-Loc* significantly improves data access performance. While *Row* performs better than *SFC* because it results in a balanced data distribution, *Pd-Loc* achieves an up to 4 times speedup over *Row*, by placing data close to the computation accessing it. Note that the speedup increases from 2 times to 4 times as the size of the accessed data grows from 30% to 50%, which indicates that the larger the data size is, the greater the potential benefits that can be gained from our adaptive data placement approach.

5.1.2 *Evaluation using the trace from the Combustion – Volume rendering workflow*

In this experiment, we emulated the dynamic data access patterns extracted from the workflow composed of the combustion simulation and the volume rendering component. In this scenario, scientists are usually interested in the scalar

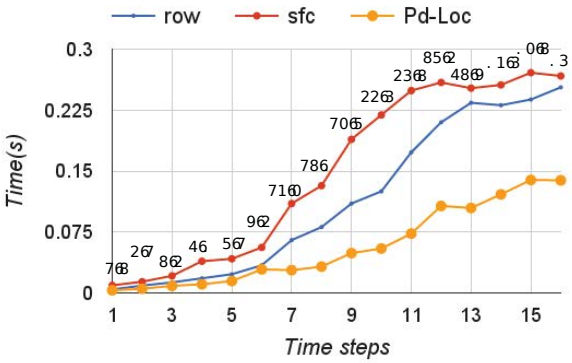


Figure 13: A comparison of data access time using different data placement approaches. The dynamic data access behaviors are based on traces from the Combustion – Volume rendering applications over 16 time steps. The sizes (GB) of the accessed data regions at each time step are also shown in the plot.

dissipation rate over time. They explore the regions to study turbulence chemistry interactions within the kernels by computing dilatation, turbulent kinetic energy, etc., within the regions, and to try to correlate them with thermo-chemical quantities.

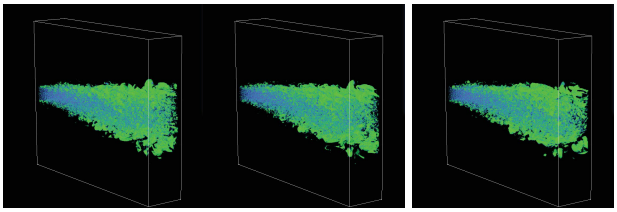
Figure 10 shows a visualization of the regions of data that are accessed at three selected time steps, showing the main flame structure of the scalar dissipation rate at those time steps. We can see that the regions are expanding from the center of the data domain outward to the surrounding area; at 9th time step, it almost spreads over the entire domain. In this case, the data access traces include 16 time steps, and the regions of data accesses grow from 0.5% to 23% of the entire data domain. Correspondingly, the accessed data size grows from 1.3GB to 59GB.

Figure 13 plots the data access time for 16 time steps using different data placement approaches. The results show that our adaptive approach (*Pd-Loc*) has the shortest data access times. This improvement is due to both, load balancing and location-aware data placement. We can observe that *Pd-Loc* achieves 3.9 and 2.5 times speed up compared to *Row* and *SFC* respectively, in the 8th time step. Note that after the 12th time step, as the data access regions spread over the entire domain, all the data placement approaches achieve a relatively balanced load across the staging servers, and results in a more stable performance. Also note that the data access time when using *Pd-Loc* is almost proportional to the data size, indicating its scalability. Specifically, as the accessed data size increased from 1.3GB to 59GB, the data access time increased from 0.003 to 0.138 seconds.

5.2 Evaluation using the production S3D analysis workflow

5.2.1 Experimental setup

In order to demonstrate the applicability of our adaptive data placement approach (*Pd-Loc*) to production application workflows, we integrated it with a real workflow composed of the S3D combustion simulation and two analysis applications – *descriptive statistics* and *feature extraction*. We then tested the performance of the integrated workflow on Titan. In this experiment, the combustion simulation



(a) TS = 1. **(b)** TS = 20. **(c)** TS = 40.

Figure 14: A visualization of the evolving data access regions – FS/S (shown in green) and FC/U (shown in blue) flow classifications for the combustion dataset that represents a lifted hydrogen jet flame. The regions of interest (shown in green) are changing slightly over time.

generated a lifted hydrogen jet flame. We describe the two analysis codes below.

Feature extraction: The volume data in the 3D lifted jet flame can be classified into 27 flow structures by computing a local rate-of-deformation tensor [7]. The two classifications, FC/U (focusing compressing unstable) and FS/S (focusing stretching stable), are the most prominent features of interests that are presumed to have a strong influence on flamelet deformation. In this experiment, we extracted the flow structure of FS/S and the combustion variables to observe interaction between the flame, jet, and boundary layer. Figure 14 presents a visualization of data regions of FS/S flow structure that are accessed at three selected time steps. The data access region, shown in green in Figure 14, changed slightly at the boundary of the flame.

Descriptive statistics: Descriptive statistics [21] is a common tool used by scientists to provide succinct summaries of trends in their data. In this experiment, we used an in-transit deployment of a scalable parallel statistic algorithm based on the VTK library [1], which is similar to one that had been used in our previous work [2].

Similar to the previous experiments, as soon as the data is generated by the simulation, it is transferred to the staging servers. Then, both analyses can access either the full data domain, or a subset of it, from the staging servers. For comparison, we used the traditional DataSpaces [9] method (labeled *DS*) based on chunking in addition to our adaptive data placement approach (*Pd-Loc*). In the traditional DataSpaces method, the size of each chunk depends on the domain decomposition for the simulation, and these chunks are placed on staging servers using a round-robin approach.

5.2.2 Experiment results

Number of cores	16512	33024	66048
No. of data producer cores	8192	16384	32768
No. of staging cores	128	256	512
No. of cores for *descriptive statistic*	4096	8192	16384
No. of cores for *feature tracking*	4096	8192	16384
Volume size	$1280 \times 640 \times 640$		
No. of variables	9		
Data size	72GB		

Table 1: This table presents the core-allocations and data volume used in the three test scenarios: 16512, 33024 and 66048 cores.

In our experimental study, we used a lifted S3D simulation of Hydrogen combustion with a grid size of $1280 \times 640 \times 640$. The core configurations, data region assignments, and data sizes are listed in Table 1.

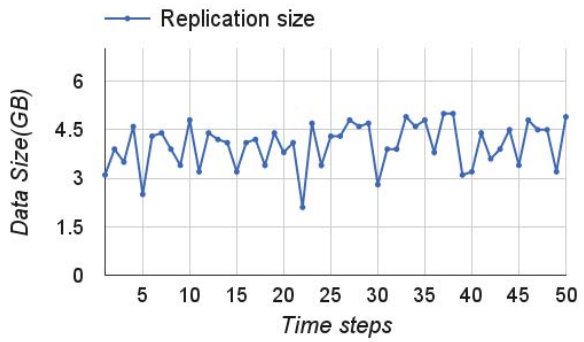


Figure 16: The size of data replicas created by the adaptive data placement over 50 time steps.

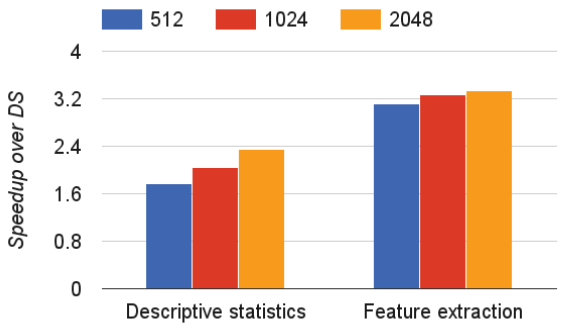


Figure 17: Speedup of *Pd-Loc* over *DS* for cumulative data access time over 50 time steps for *feature extraction* and *descriptive statistics* analyses using increasing number of nodes.

Different from the previous experiments, this experiment involves two data consumer applications running concurrently and accessing data from the staging area. The *Descriptive statistics* always accesses the full data domain, leading to static data access patterns. In contrast, *feature extraction* accesses a subdomain, which changes over time, leading to dynamic data access patterns. Our data placement has to adapt to the combined dynamic data access patterns of the two data consumers to improve data access performance for both analyses.

For this experiment we used the first core configuration listed in Table 1. We measured the data access time for each analysis code separately and plotted the results over 50 time steps in Figure 15. As we expected, except for the first time step, our adaptive data placement (*Pd-Loc*) outperforms the traditional *DS* approach in terms of data access performance for both analyses. We can observe that the data access time for *feature extraction* varied more significantly than that for *descriptive statistics* for both data placement approaches. This variability was caused by the changing data access patterns exhibited by *feature extraction*. Because of the adaptability of our approach, it achieved much better performance, which is consistent with the results in the previous experiments. There is a slight fluctuation in the results using our adaptive approach, which can be attributed to the imperfect prediction of the irregular data access patterns by our prediction mechanism, resulting in increased data access times. However, our adaptive approach still achieves up to a 3 times speedup as compared to the *DS* approach.

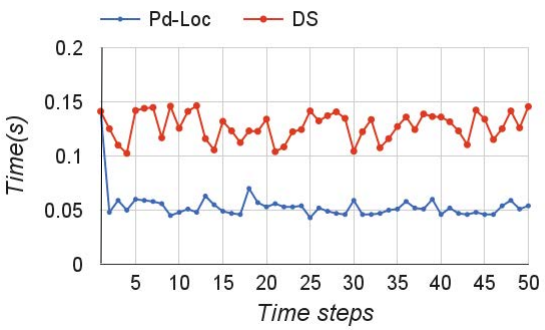
In order to satisfy the requirements of both the analyses, our adaptive approach created replicas of the data as explained in Section 3. We measured the size of these replicas for each time step, which is plotted in Figure 16. The data region that is concurrently accessed by both the analyses in this experiment is the same as the region accessed by *feature extraction*, which is around 9GB in each time step. We observe that the maximum replication size is 5.1GB, and the minimum size is 2.3GB, which is only 25% of the overlapped data region. This experiment demonstrates that our approach uses storage efficiently.

Strong Scaling: We also performed strong scaling experiments to evaluate the scalability of our approach. We kept the data configuration the same, but scaled both simulation and analysis applications to run on 512, 1024 and 2048 nodes. Correspondingly, the simulation ran on 8k, 16k and 32k cores, while each analysis application ran on 4k, 8k and 16k cores, respectively. We evaluated the cumulative data access time across 50 time steps for both *feature extraction* and *descriptive statistics*, and plotted the speedup of *Pd-Loc* over *DS* in Figure 17. We observe that our adaptive data placement (*Pd-Loc*) yields an increasing performance improvement over the *DS* approach as the number of nodes scales up. This improvement is mainly a result of the location-aware mechanism. At smaller node counts, the impact of network contention/congestion is not as significant as it is in a larger network. However, our experiments conclude that the our adaptive data placement approach shows appropriate scaling for large node counts.

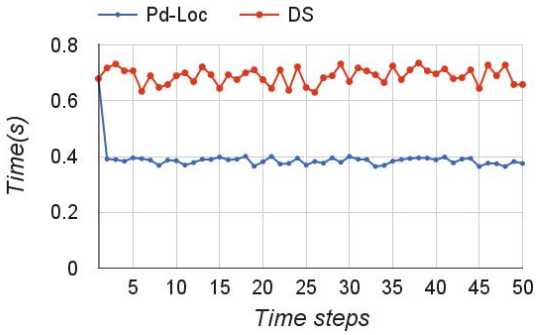
System overheads: We also evaluated the system overheads, which are mainly caused by the runtime data placement in the staging area. We have identified three primary sources of overheads, as follows: (1) time to acquire the dynamic data access patterns, (2) time to collect data requirement information from staging servers, and (3) time to distribute the data placement decision to the staging servers. The total cumulative overheads over 50 time steps for 128, 256 and 512 cores running the staging servers are 0.031, 0.087 and 0.26 seconds, respectively. When compared with the end-to-end time of the simulation, which is often on the order of hundreds of seconds long, these data placement overheads can be considered negligible. In addition, the adaptive data placement can be performed concurrently with the application's computations, and therefore, it would not have impact on the overall end-to-end performance of the workflow.

5.3 Discussion

The efficiency of our adaptive data placement approach relies on the accuracy of anticipated dynamic data access patterns. As demonstrated in the experiments, this approach works well for applications with repetitive data access behaviors since we can accurately capture data access patterns at runtime. For other applications with irregular data access patterns, our prediction mechanism, like all prediction strategies, has its limitations. For example, it may not accurately predict data access patterns that can be used for data placement. However, we have demonstrated that our adaptive data placement approach, based on runtime prediction, can achieve considerable improvement in data access performance for data access patterns that are changing smoothly



(a) The data access performance for *feature extraction*.



(b) The data access performance for *descriptive statistics*.

Figure 15: A comparison of the data access performance over 50 time steps for the two analyses that are part of the S3D-analysis workflow.

and following a fairly predictable path. It has been shown that such access patterns exist for a variety of scientific simulations and analyses codes, such as feature tracking and data trajectories visualization [27], which indicates that our approach can be effectively applied to a wide range of existing scientific applications.

Worst-case analysis: The worst case happens when the predicted access pattern is completely incorrect, i.e., all the data is placed on the staging servers that are 'farthest' from the computational nodes that will access the data. In this case, retrieving data from the staging area would take the maximum time. As a result, the worst-case performance of our approach is the same as that of other static approaches, such as *row* placement. The only additional penalty in such a worst-case when using our approach is the overheads due to data placement and potential data replications. Based on our evaluation, these overheads are relatively small compared to the overall execution time, and data replications are only created when needed.

Handling other network topologies: Our research presented in this paper has focused on the torus/mesh network, which is used in current high-end systems. In case of the dragonfly network architecture, which potentially has less network contention, reducing data transfer distance might not result in a significant improvement in the read performance. However, in a dragonfly network, specific communication pattern can overload certain links and increase network congestion, resulting in high communication latency [13]. Therefore, we need to explore different approaches for reducing data access latency with our framework. For example, we can utilize network topology information and adapt data placement to use alternate communication pattern in order to avoid the network bottleneck. We plan to explore these and other options in our future work.

6. RELATED WORK

Data placement for scientific workflows: Several research efforts have explored different adaptive techniques for data placement for scientific workflows. For example, Stork [17] is a data placement service on grid systems and uses an adaptive network protocol to move data efficiently. Popular scientific workflow management systems, such as Kepler [19], Pegasus [8] and DAGMan [20], also include data management strategies. The data is stored in files and is moving among applications automated through workflow engines. Some of these systems rely on users to specify the data management steps using workflow scripts, which describe the sequence of execution of the composed applications, the order of exchange of data files. Therefore, these systems have a limited ability to handle dynamic scientific workflows, in which the interactions and data exchange patterns between applications are not know a priori. Other systems provide automated mechanisms to manage data transfers. These systems use the existence of files (or some other form of output) to detect the end of a write phase and to trigger data transfers. However, due to the nature of parallel file systems such as Lustre, such runtime monitoring and 'notification' can cause contention and slow downs in the IO system thereby impacting the performance of the applications.

In addition, using files as the data exchange mechanism often relies on specialized shared, parallel file systems (PFS). Some research efforts such as [28, 25] focus on I/O optimization by reorganizing data inside the file-system to improve performance. Such data reorganization would have little impact on the workflow communication itself as the PFS typically has dedicated resources such as network bandwidth. However, for the staging-based in-situ/in-transit workflows targeted in this paper, both staging servers and applications run on the same system and share network resources. As a result, we have to reduce data placement and access overheads to satisfy the strict performance and overheads requirements.

In-staging data placement: A number of data-staging frameworks have been developed in recent years, such as DataSpaces [9] /ActiveSpaces [10] and PreDatA [31], to support scientific workflows using memory within a data staging area for runtime data coupling and exchange. However, these frameworks do not optimize the placement of data in the staging area to improve data access performance, especially for workflows with dynamic data access patterns. For example, DataSpaces places data using a hash function based on the MPI rank without considering application data access patterns. Similarly, existing researches have also focused on using data staging for data preparation operations such as filtering [31] and indexing and querying [24], to optimize subsequent data analysis steps within a workflow. However, none of these efforts address the optimization of data placement within the staging area.

Techniques for optimizing data placement: Developing storage systems geared towards specific access pat-

terns is not uncommon. For example, GFS [11] is optimized for large data sets and append accesses; EDO [25] explores SFC-based data organization to improve the read performance for its targeted scientific applications. However, the optimization techniques in these systems are designed for specific access patterns, and can become ineffective when applications within a workflow have distinct and dynamic data access patterns. PDLA [28] optimizes data layout based on I/O behaviors captured in traces, to improve the data access performance of parallel I/O systems. Our work similarly captures application access patterns at runtime. However, in contrast, it places data to distributed in-memory storage in a topology-aware manner. Several research efforts have used replication to optimize access for parallel/distributed storage systems, such as HDFS [4] and GPFS-SNC [12]. In these approaches, multiple copies of data files are placed at different locations, and the systems [4, 12] select the 'best' (in terms of access cost) replica for accesses. Our data placement differs in that it dynamically creates replicas at runtime based on current data access patterns and memory resource availability within the staging area.

7. CONCLUSION

In this paper, we proposed an adaptive data placement approach to improve the data access performance for staging-based scientific workflows running on large-scale systems. Our approach leverages application-specific data access patterns and system network topology to dynamically optimize the data placement in the staging area. It focuses on making runtime data placement adaptation on-the-fly, while data is being transferred from the data producer application to the staging area. This paper also presented the design and implementation of a runtime system that realizes our data placement approach. We have deployed the runtime on the Titan Cray XK7 and experimentally evaluated it using a production S3D combustion simulation workflow. The experiments demonstrate that our data placement approach effectively adapts to dynamic data access patterns and achieves up to a 3 times speedup over other static approach in terms of data access times.

Our future work will extend the data placement approach for other network topologies such as dragonfly, and for other data representations in scientific applications such as graphs. We will also explore the optimization of data placement by taking into consideration the energy/power efficiency and power/performance trade-offs.

8. ACKNOWLEDGMENTS

The research presented in this work is supported in part by National Science Foundation (NSF) via grants numbers ACI 1339036, ACI 1310283, CNS 1305375, and DMS 1228203, by the Office of Advanced Scientific Computing Research, Office of Science, of the US Department of Energy through the SciDAC Institute for Scalable Data Management, Analysis and Visualization (SDAV) under award number DE-SC0007455, RSVP award via subcontract number 4000126989 from UT Battelle, the ASCR and FES Partnership for Edge Physics Simulations (EPSI) under award number DE-FG02-06ER54857, and the ExaCT Combustion Co-Design Center via subcontract number 4000110839 from UT Battelle. The research at Rutgers was conducted as part of the Rutgers Discovery Informatics Institute (RDI^2).

9. REFERENCES

[1] Vtk doxygen documentation. http://www.vtk.org/doc/nightly/html.

[2] J. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–9, Nov 2012.

[3] A. Bhatele and L. Kale. An evaluative study on the effect of contention on message latencies in large supercomputers. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8, May 2009.

[4] D. Borthakur. Hdfs architecture guide. 2008.

[5] P. Bremer, E. Bringa, M. Duchaineau, A. Gyulassy, D. Laney, A. Mascarenhas, and V. Pascucci. Topological feature extraction and tracking. In *Journal of Physics: Conference Series*, volume 78, page 012007. IOP Publishing, 2007.

[6] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo. Terascale direct numerical simulations of turbulent combustion using s3d. *Computational Science & Discovery*, 2009.

[7] M. S. Chong, A. E. Perry, and B. J. Cantwell. A general classification of three-dimensional flow fields. *Physics of Fluids A*, 2(5):765–777, 1990.

[8] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. hui Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid, 2004.

[9] C. Docan, M. Parashar, and S. Klasky. Dataspaces: an interaction and coordination framework for coupled simulation workflows. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 25–36, 2010.

[10] C. Docan, F. Zhang, T. Jin, H. Bui, Q. Sun, J. Cummings, N. Podhorszki, S. Klasky, and M. Parashar. Activespaces: Exploring dynamic code deployment for extreme scale data processing. Wiley Online Library, 2014.

[11] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. volume 37, Oct. 2003.

[12] K. Gupta, R. Jain, I. Koltsidas, H. Pucha, P. Sarkar, M. Seaman, and D. Subhraveti. Gpfs-snc: An enterprise storage framework for virtual-machine clouds. *IBM Journal of Research and Development*, 2011.

[13] N. Jain, A. Bhatele, X. Ni, N. J. Wright, and L. V. Kale. Maximizing throughput on a dragonfly network. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 336–347, 2014.

[14] T. Jin, F. Zhang, Q. Sun, H. Bui, N. Podhorszki, S. Klasky, H. Kolla, J. Chen, R. Hager, C.-S. Chang, et al. Poster: Leveraging deep memory hierarchies for data staging in coupled data-intensive simulation workflows. In *Cluster Computing (CLUSTER), 2014*

IEEE International Conference on, 2014.

[15] 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*, pages 1033–1042, May 2015.

[16] K. Kandalla, H. Subramoni, A. Vishnu, and D. Panda. Designing topology-aware collective communication algorithms for large scale infiniband clusters: Case studies with scatter and gather. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, April 2010.

[17] T. Kosar and M. Livny. Stork: making data placement a first class citizen in the grid. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 342–349, 2004.

[18] J. Lofstead, M. Polte, G. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf, and Q. Liu. Six degrees of scientific data: Reading patterns for extreme scale science io. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11, pages 49–60.

[19] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065.

[20] G. Malewicz, I. Foster, A. Rosenberg, and M. Wilde. A tool for prioritizing dagman jobs and its evaluation. *Journal of Grid Computing*, 5(2):197–212, 2007.

[21] P. Pebay, D. Thompson, and J. Bennett. Computing contingency statistics in parallel: Design trade-offs and limiting cases. In *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*, pages 156–165, Sept 2010.

[22] G. Ross and R. Soland. A branch and bound algorithm for the generalized assignment problem. *Mathematical Programming*, pages 91–103, 1975.

[23] S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *Data Engineering, 1994. Proceedings.10th International Conference*, pages 328–336, Feb 1994.

[24] Q. Sun, F. Zhang, T. Jin, H. Bui, K. Wu, A. Shoshani, H. Kolla, S. Klasky, J. Chen, and M. Parashar. Scalable run-time data indexing and querying for scientific simulations. In *Big Data Analytics: Challenges and Opportunities (BDAC-14) Workshop at Supercomputing Conference*, SC '14, November 2014.

[25] Y. Tian, S. Klasky, H. Abbasi, J. Lofstead, R. Grout, N. Podhorszki, Q. Liu, Y. Wang, and W. Yu. Edo: Improving read performance for scientific applications through elastic data organization. In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 93–102, Sept 2011.

[26] L. Vervisch, E. Bidaux, K. N. C. Bray, and W. Kollmann. Surface density function in premixed turbulent combustion modeling, similarities between probability density function and flame surface approaches. *Physics of Fluids (1994-present)*, 7(10):2496–2503, 1995.

[27] J. Wei, H. Yu, R. Grout, J. Chen, and K.-L. Ma. Dual space analysis of turbulent combustion particle data. In *Pacific Visualization Symposium (PacificVis), 2011 IEEE*, pages 91–98, March 2011.

[28] Y. Yin, J. Li, J. He, X.-H. Sun, and R. Thakur. Pattern-direct and layout-aware replication scheme for parallel i/o systems. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 345–356, May 2013.

[29] H. Yu, C. Wang, R. W. Grout, J. H. Chen, and K.-L. Ma. In situ visualization for large-scale combustion simulations. *IEEE Computer Graphics and Applications*, (3):45–57, 2010.

[30] 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.

[31] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, Q. Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. Predata - preparatory data analytics on peta-scale machines. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12, April 2010.