A Scalable Messaging System for Accelerating Discovery from Large Scale Scientific Simulations

Tong Jin, Fan Zhang, Manish Parashar
The NSF Center for Cloud and Autonomic Computing
Rutgers University, Piscataway NJ, USA
{tjin,zhangfan,parashar}@cac.rutgers.edu

Scott Klasky, Norbert Podhorszki, Hasan Abbasi
Oak Ridge National Laboratory
P.O. Box 2008, Oak Ridge, TN, 37831, USA
{klasky,pnorbert,habbasi}@ornl.gov

Abstract—**Emerging scientific and engineering simulations running at scale on leadership-class High End Computing (HEC) environments are producing large volumes of data, which has to be transported and analyzed before any insights can result from these simulations. The complexity and cost (in terms of time and energy) associated with managing and analyzing this data have become significant challenges, and are limiting the impact of these simulations. Recently, data-staging approaches along with in-situ and in-transit analytics have been proposed to address these challenges by offloading I/O and/or moving data processing closer to the data. However, scientists continue to be overwhelmed by the large data volumes and data rates.**

In this paper we address this latter challenge. Specifically, we propose a highly scalable and low-overhead associative messaging framework that runs on the data staging resources within the HEC platform, and builds on the staging-based online in-situ/in-transit analytics to provide publish/subscribe/notification-type messaging patterns to the scientist. Rather than having to ingest and inspect the data volumes, this messaging system allows scientists to (1) dynamically subscribe to data events of interest, e.g., simple data values or a complex function or simple reduction (*max()/min()/avg()*) of the data values in a certain region of the application domain is greater/less than a threshold value, or certain spatial/temporal data features or data patterns are detected; (2) define customized in-situ/in-transit actions that are triggered based on the events, such as data visualization or transformation; and (3) get notified when these events occur. The key contribution of this paper is a design and implementation that can support such a messaging abstraction at scale on high-end computing (HEC) systems with minimal overheads. We have implemented and deployed the messaging system on the Jaguar Cray XK6 machines at Oak Ridge National Laboratory and the Lonestar system at the Texas Advanced Computing Center (TACC), and we present the experimental performance evaluation using these HEC platforms in the paper.

Keywords-**associative messaging system, publish/subscribe, in-situ/in-transit analytics, data staging.**

I. Introduction

Emerging scientific and engineering simulations running at scale on leadership-class High End Computing (HEC) environment have the potential for leading to dramatic insights into complex physical phenomena by enabling increasingly accurate solutions to realistic models. At the same time, these simulations are also generating large amounts of data that has to be transported and analyzed before this potential can be realized. This increase in data volume presents significant challenges in terms of (1) the costs (time and energy) required to transport the data as well as (2) the complexity of translating the data into insights, both of which are significantly reducing the productivity of the scientists and the impact of the simulations. For example, turbulent combustion direct numerical simulations (DNS) attempt to resolve intermittent phenomena that occur on the order of 10 simulation time steps. However, in order to maintain I/O overheads and data management complexities within a reasonable level, data is only extracted and analyzed every 400 time steps, and, as a result, some intermittent phenomena are lost. Furthermore, the analysis is typically done offline as a post-processing step.

Several research efforts have tried to address these challenges and explored techniques for high-throughput data transfers and efficient data management with low application overhead. One promising approach is to offload expensive I/O and data processing operations from the computing cores to the *staging area* that comprises of a smaller set of dedicated cores, typically on the same HEC system. For example, our work with DataSpaces [1] uses this approach and essentially builds a distributed shared space abstractions on the staging cores, which can be associatively accessed by all applications/services that are parts of the simulation workflow, (i.e., simulation codes, coupling methods, analysis and visualization codes, etc.) and enables them to asynchronously insert and retrieve data to/from the shared space. This class of solutions primarily focuses on the fast and asynchronous movement of data from/to simulation cores to reduce the impact of expensive I/O operations on the simulations.

Furthermore, *in-situ* and *in-transit* analytics approaches have also been proposed that use compute and memory resources at the stating area to process the data closer to the compute cores. For example, projects such as DataSpaces [1]/ActiveSpaces [2], DataStager [3]/PreData [4] and Glean [5] have explored how staging resources can be used to add value to the data path by executing parts of the application's data analysis pipeline.

While techniques such as data staging, I/O offloading, and *in-situ* and *in-transit* analytics have helped address the challenges and costs associated with transporting, managing and processing the large volumes of data, the second challenge – the complexity of translating data into insights remains largely unaddressed and scientists continue to be overwhelmed by the large data volumes and data rates – scientists have to process all the data and/or data products to detect the events of interest that they may want to explore further.

In this paper we address this second challenge and explore how the staging resources and the data staged there along with *in-situ* and *in-transit* analytics can be leveraged to automate the detection of event of interest and to improve the productivity of the scientists. Specifically, we propose a highly scalable and low-overhead associative messaging system that runs on the staging resources within the HEC platform and leverages the user-defined online in-situ/in-transit analytics there, and provides publish/subscribe/notification-type messaging patterns [6] to the scientists. The messaging system builds on DataSpaces [1] and ActiveSpaces [2], and allows scientists to (1) dynamically subscribe to data events of interest possibly in regions of interest, (2) define actions that are triggered based on the events, and (3) get notified when these events occur. Both, events and triggered actions can be based on *in-situ* and *in-transit* analytics operations. For example, the registered data event may specify that a function or simple reduction (*max()/min()/avg()*) of the data values in a certain region of the application domain is greater/less than a threshold value, or a certain spatial/temporal data feature or data pattern is detected; and the resulting actions include users getting notified and user-defined *in-situ/in-transit* actions, e.g., visualization or writing the target data to persistent storage, being triggered at the staging nodes.

Such a messaging system provides several advantages. It enables scientists or scientific applications to dynamically formulate and asynchronously submit subscriptions for data patterns of interest in a non-blocking manner, and to get notified when such events occur. It thus avoids the overheads of continuously monitoring data-availability and retrieving data to detect events of interest. The messaging framework also enables the online screening of data, enabling scientists to quickly discover coarse-grained data characteristics and decide whether subsequent data movement and further detailed analysis are necessary. Furthermore, customized actions can be triggered by the occurrence of events of interest, and enable flexible user-defined *in-situ/in-transit* processing within the staging area performed concurrently with the running simulations.

Note that, while publish/subscribe messaging systems have been explored in the past to support a range of applications [7]–[11], a key contribution of this work is a highly scalable design and implementation targeted at HEC platforms that can run along with high-performance large scale simulations. Our messaging platform builds on data staging techniques and in-situ/in-transit analytics to support higher-level abstractions (e.g., features, data patterns) at which subscriptions can be defined, as well as support concurrently performed actions that can be triggered by events of interest. This minimizes the impact on the simulations and can accelerate scientific discovery and the scientists' productivity.

We have implemented and deployed the messaging system on the Jaguar Cray XK6 system with Gemini interconnect at Oak Ridge National Laboratory and the Lonestar system at the Texas Advanced Computing Center (TACC), and we present the experimental performance evaluation using these HEC platforms in the paper.

The rest of the paper is organized as follows. Section II presents two motivating application workflow scenarios. Section III presents the overall architecture and design of the proposed messaging system and describes the semantics of the messaging abstractions provided. This section also provides an overview of DataSpaces. Section IV describes the implementation, and Section V presents the experimental evaluation. Section VI presents the related work, and Section VII concludes the paper.

II. Background

In this section, we present two application scenarios that motivate the design and implementation of our system.

A. *Scientific Simulation and Analytics*

The "simulation-analytics" workflow is widely used in different research areas and scientific applications for scientific discovery. For example, the study of turbulent combustion [12] is extremely significant in terms of the development of new efficient combustion technologies and addresses the challenges related to climate change and air pollution. Direct numerical simulation (DNS) [13] can provide relevant statistical information to efficiently help the modeling of turbulent combustion. While the simulation data is being generated, subsequent analytics such as descriptive statistic analysis, visualization, and topology-based analysis [14] assist scientists in gaining insights and understanding causalities behind the physical phenomena.

Previous work on descriptive statistic analysis [15] has shown great interests in capturing and tracking flow features and their statistics together with their correlation with associated scalar quantities, e.g., temperature or species concentrations. For example, based on the features tracked, appropriate computations are triggered and/or a collection of meta-data is generated, which can be orders of magnitude smaller than the original simulation data. This pattern can also be used for topology analysis and visualization in combustion applications. In this scenario, only if features of interest are detected, appropriate computational analysis and visualization actions need to be taken, avoiding the heavy I/O overhead in traditional approach that dumps all the data and post-processes them offline.

The presented messaging system allows scientists of registering features of interest and defining actions when the feature is detected, which monitors the simulation and automates the analytics dynamically and efficiently.

B. *Online Data Monitoring and Adaptations*

As an explicit example, consider the petascale fusion application being developed by researchers at Oak Ridge National Laboratory. This scientific investigation attempts to better understand the stability of contained fusion plasma, which involves large (independently developed) codes that implement complex mathematical models for the core and edge of the plasma, and the runtime couplings of these codes.

This application requires continuous monitoring of the coupled simulations as they run to ascertain whether certain events (e.g., elm event) have occurred and that the results of the coupling are scientifically meaningful, and to adapt the simulations accordingly when such events are detected. The presented messaging framework can effectively support such online data monitoring and automatic adaptations without pausing/restarting the simulation.

III. System Architecture

A. *Overview of DataSpaces*

In this research work, our messaging framework is built on the top of the DataSpaces [1] framework, which is a distributed semantically specialized shared space abstraction that is accessible by all the components and services in an application workflow. DataSpaces mediates communications amongst applications and provides them with the data services such as asynchronous data insertion and retrieval. For instance, the run-time simulation may easily output data into DataSpaces by calling *put()* operator; and the visualization process running on another system could fetch the data objects stored in DataSpaces by calling *get()*. The internal data management mechanisms of DataSpaces ensure the scalability of distributed data storage and lookup across the staging nodes.

DataSpaces consists of two basic components, a DataSpaces server that runs on the staging area and a DataSpaces client, which is integrated with the applications on the computing nodes. The DataSpaces server utilizes the memory space of all the staging nodes to build a distributed *data storage* layer, which temporarily stores and maintains the data objects aggregated from user applications. Beyond the storage layer, DataSpaces implements data services like *data lookup* and *query engine* to refer and access the data objects across the server nodes. Space-filling curve (SFC) indexing is used on the server side, which maps the multidimensional application domain to a linear index and constructs a distributed hash table (DHT) to manage the storage location information of data objects. The SFC indexing and DHT serve the *data lookup* module, enabling quick and flexible data lookups. Also, it allows the overall DataSpaces framework to be scalable, flexible and handle large volume of distributed data storage. The *query engine* benefits from this indexing mechanism and enables fast and efficient data insertion and retrieval.

The DataSpaces client is a lightweight component that integrates with the applications and exposes the API to access and leverage the data services supplied from the DataSpaces server side. It runs on application computing nodes and allows the application to query the data objects from the staging server or vice versa. If the data objects are located on multiple server nodes, it transparently dissembles the data query and disseminates the sub-queries to DataSpaces servers, and then locally assembles the final results.

DataSpaces is often implemented on top of underlying data communication layer, *e.g. DART* [16], with remote direct memory access (RDMA) to enable asynchronous in-memory data transfer. This asynchronous computation pass-by memory-to-memory exchange mechanism avoids the latency of the parallel file-systems caused by concurrent accesses by multiple applications, so as to accelerate the data transfer with less variability than using persistent storage systems like files or databases.

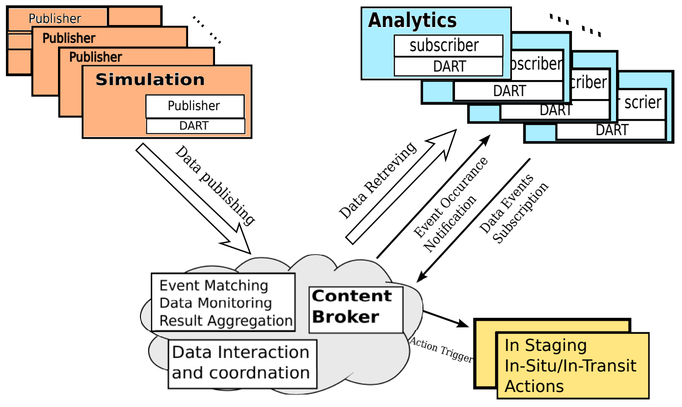


Fig. 1. System component and semantic workflows.

B. *System Components*

We implemented a content-based *Publish-Subscribe* messaging layer on the top of DataSpaces to support more flexible non-blocking data querying, event registration, and other relevant services. Our distributed messaging framework derives from DataSpaces and extends its main components: a client component (publisher or subscriber) and a server (broker) component (Figure 1). Different semantical interfaces are provided for each component.

Client Component: The Client is a lightweight component that integrates with upper layer applications and utilizes the data services through exposed programming API. The client could run either as a publisher to spawn simulation data into server component, or a subscriber to submit specific data events of interests and corresponding actions to the server side. Not only the value-based events like a certain range of data value could be defined here, feature-based data events are also supported, *e.g.* boundary coordinates of a data block, thresholding of data characteristics like *max()/min()/svg()*.

Server Component: The server is a distributed component based on existing DataSpaces services. It provides a temporary in-memory storage space and enables the data services to interact with this space, such as data inserting and retrieving, and data indexing and lookup. It also acts as an intermediate message broker which provides extended new data services for real time data events detection, which includes publish/-subscribe management, event matching, data execution, and notification dissemination. These staging nodes keep matching the incoming live data against a set of subscriptions containing the data events, and trigger the relevant actions based on the event. The server component consists of three key layers to support these services: a data movement and communication

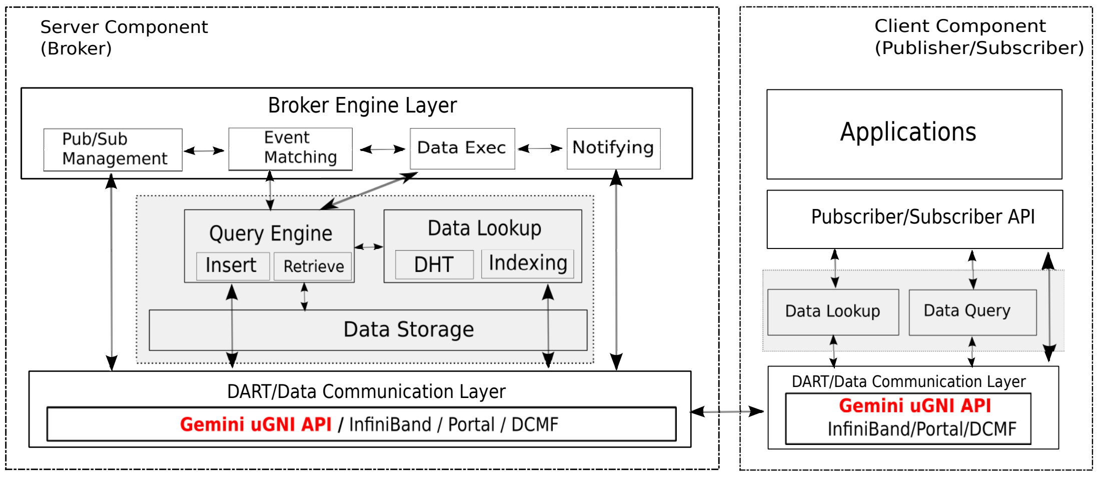


Fig. 2. Layered system architecture (shaded area is DataSpaces layer).

layer, a Dataspaces layer, and a broker engine layer, as described below.

C. *Layered Architecture*

Data Movement and Communication Layer: This layer builds on DART [16], which is an asynchronous communication and data transport abstraction based on RDMA one-sided communication. DART supports asynchronous message passing and data transport between the parallel simulation nodes and the staging area. It provides semantical services such as node registration/deregistration, message passing, data transfer, event notification and processing. DART can be easily ported onto different communication fabrics with scalable performance. A significant contribution in this paper is that we implement the DART functionality on the Gemini network of the Cray's XE/XK systems.

Dataspace Layer: This layer directly uses the DataSpaces [1] framework, which is a distributed interaction and coordination service. It provides efficient internal services like in-memory data storage, data lookup and data querying for upper layer usage. DataSpace implements a scalable, semantically specialized shared space abstraction that is accessible by all the components and services in an application workflow. Due to the asynchronous, low-overhead, memory-to-memory data transport provided by the data movement and communication layer, it allows applications to overlap interactions and data transfers with computation, and to reduce the I/O overheads by offloading data operations to the staging area. In this work, we use its data query, lookup, and storage modules to support upper layer functionalities.

Broker engine layer: Broker engine layer provides the central semantics and mechanisms of publish/subscribe management, event matching, action triggering and distributed query merging across the servers. Essentially, this layer offers four types of modules, as shown in Figure 2: (a) *Pub/sub management module* caches the coming publishing/subscriptions and removes them when they are processed. (b) *Event matching module decodes* the data events from subscriptions, tracks properties of online data sets from publishers and matches data events, and triggers pre-defined actions. The basic descriptive statistic functions like *min()/max()/avg()* are embedded in this module. (c) *Data execution module* defines the *in-situ/in-transit* actions triggered on the occurrence of data events, like data transforming, content filtering, visualization, and other customized functions. (d) *Notifying module* performs notification if requested. The mechanisms and detailed implementation of these functions are presented in the next section.

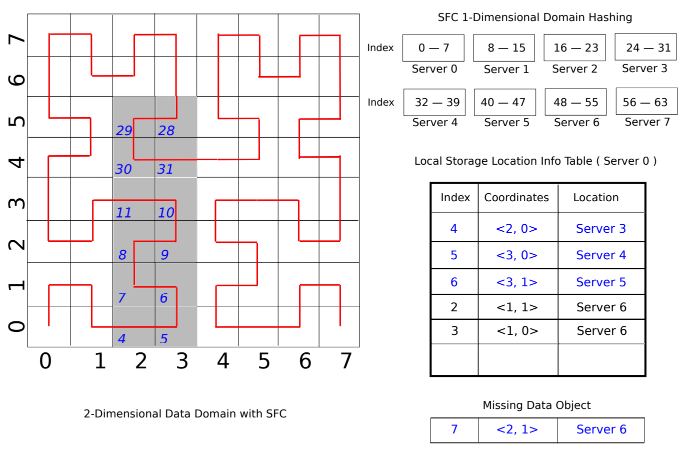
IV. Implementation

In this section, we present the design and implementation of the central mechanisms underlying our messaging system. These mechanisms effectively support the functionality and performance of the system, and ensures its scalability and dynamic operation.

A. *Dynamic Subscription Management*

The data services offered by our messaging system are non-blocking, and client applications can asynchronously register data events of interest related to any data region of the computation domain. On receiving subscriptions from the clients, the server must manage and appropriately process subscriptions and events. The following mechanisms are implemented to support flexible and dynamic subscription/event management.

Subscription Caching: Multiple subscriptions with unique identifiers are temporarily cached on the server side in a *Subscription Queue*. When data is generated, the server nodes monitor the completeness of associated data objects on the basis of the cached local data events, and determine the subsequent event matching. After the server successfully processes the data event and notifies the user, the associated subscription will be removed from *Subscription Queue*, and moved to a *Hit Queue* that contains the most recently processed events, in order to reduce the processing cost of same events and improve system performance.


SFC 1-Dimensional Domain Hashing
2-Dimensional Data Domain with SFC
0
1
2
3
4
5
6
7
29
28
30
31
11
10
8
9
7
6
4
5

Index	0 — 7	8 — 15	16 — 23	24 — 31
	Server 0	Server 1	Server 2	Server 3
Index	32 — 39	40 — 47	48 — 55	56 — 63
	Server 4	Server 5	Server 6	Server 7

Local Storage Location Info Table (Server 0)

Index	Coordinates	Location
4	<2, 0>	Server 3
5	<3, 0>	Server 4
6	<3, 1>	Server 5
2	<1, 1>	Server 6
3	<1, 0>	Server 6

Missing Data Object

7	<2, 1>	Server 6

Fig. 3. 2-Dimensional Data Domain Example.

SFC Indexing for Data Completion Detection: Space Filling Curve (SFC) indexing is used to monitor the completeness of requested data in a region of interest on the server side. When a new data object is inserted into the staging server, its storage location information will be updated to a hashed server node based on the index of this new data object and the distributed hash table (DHT) provided by DataSpaces layer [1]. Then, if it intersects the requested data region, the system monitors the data completeness – whether all the registered data subsets in the region of interest have been completely put and stored on the server side.

The process of data completeness monitor has three steps. Firstly, each staging server uses SFC to map the subscriber-requested multidimensional region of interest to a linear index over the entire application domain, and calculates the total number of associated data elements that are within the requested region and also hashed to it. Each staging server node provides the storage location information for its associated data elements. Secondly, the staging server node counts the number of associated data elements that have been successfully inserted into the staging area, by looking up the data storage location information in the table. Finally, if the data element numbers from previous two steps are equivalent, which means the anticipated data of interest has already been stored in the staging nodes completely, the subsequent event matching is triggered. Otherwise, the monitoring process repeats with the insertion of a new data object. For example, in Figure 3, the data region of interest falls into the shaded area, whose storage location information is hashed to *Server 0*, *Server 1*, and *Server 3*. *Server 0* checks the completeness of its associated data elements following the monitoring process described above, and calculates the number of requested data elements as 4 in the first step. After looking up the local data storage location information table, *Server 0* counts the total number of inserted data elements as only 3, indicating that the requested data region is incomplete. If the missing data object *(2,1)* is inserted, the subsequent registered event will be processed.

This monitoring approach enables asynchronous data event registration and dynamic data insertion.

B. Binary Tree Based In-network Aggregation

In many cases, the data associated with a user subscription may be distributed across different staging server nodes. It is typically not reasonable to reassemble the scattered data subsets together at a single node and then process it as a whole, since it would result in large data movement among the nodes as well as redundant memory copy operations. An alternate approach is to query/analyze the distributed data subsets in parallel and in-situ, and then aggregate the result at the client. However, such a method would result in a large number of interactions between the servers and the clients, especially when the data is distributed across many server nodes, and the clients would need to have sufficient capabilities to assemble the intermediate results and compute the final result.

In our messaging framework, we leverage the computing capabilities of the staging servers to compute the result of a query in-transit using a hierarchical divide-and-conquer approach – the data subsets matching a subscription are analyzed in parallel at each server node and the intermediate results are merged and aggregated along a binary tree path as they are routed towards a dynamically selected server node. This node at the root of the tree then summarizes the result, trigger the user-specified actions, and notifies the client.

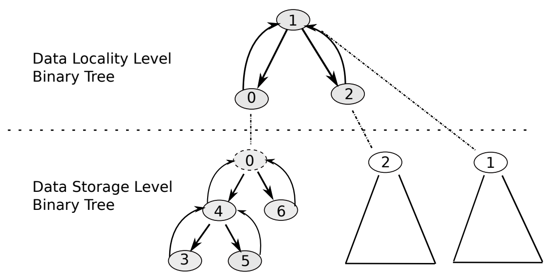


Fig. 4. Binary Aggregation Tree.

For each subscription, we thus construct a temporarily binary tree, which is then used for in-network aggregation. The binary tree is composed of staging server nodes with data that matches a subscription. When a subscription is registered, the server node looks up the *SFC DHT* and *data storage hash table* for all the staging severs nodes that are associated with the query, and constructs the binary tree, which is then use to aggregate results generated from data event matching the subscription.

Figure 4 illustrates the construction of the binary tree and the recursive aggregation of results for the example presented in Figure 3. In this figure, the user subscribes for the minimum data value inside the shaded region. Looking up the SFC DHT, *Server 0*, *Server 1*, and *Server 2* are identified as storing the locality information of data storage associated with the region of interest. These servers are then used to construct the first level of the binary tree, as shown in Figure 4. Then, each of them refers to its own local data storage location information tables and find all the staging server nodes that store the associated data subsets, then construct the second level binary tree respectively and disseminate it with data event request to the associated data storage servers.

In the example shown in Figure 3, *Server 0* finds *Server 3, Server 4, Server 5, and Server 6* as the nodes at the second-level of the tree. The data reduction required to compute the *minimum value* requested by the subscription can be simply computed in a hierarchically manner along this binary tree. Compared with direct *all-to-one* aggregation method, this in-transit binary aggregation tree balances the number of messages passing through each node, especially when the same cluster of nodes are simultaneously involved in processing multiple events. This has been validated in our baseline experiments.

C. uGNI-based Asynchronous Data Transfer

One of the key contribution of our work is that we have implemented and deployed our messaging system on top of the Gemini interconnect network using the DART data communication layer. The Gemini network provides user Generic Network Interface (uGNI) [17] as its low-level interface. User-space communications in uGNI is supported by a set of data transfer functions using the *Fast Memory Access* (FMA) and the *Block Transfer Engine* (BTE) mechanisms. To ensure efficiency and scalability, DART dynamically adapts which mechanism is used on the basis of data size. For small message sizes, DART uses the GNI *Short Message* (SMSG) mechanism, which leverages FMA and allows for direct OS-bypass achieving the low latencies and high message rates. For large data transfers, the BTE memory operations (RDMA Get and RDMA Put) are used to achieve lower performance overhead and better computation-communication overlap. The completion of an FMA or BTE transaction generates a corresponding event notification at both the source and destination of the data transfer, allowing DART to track the status of a transaction and schedule related data analysis operations.

V. Experiment and Evaluation

We have implemented and tested our system prototype on both Jaguar Cray XK6 system at Oak Ridge National Laboratory's National Center for Computational Sciences (NCCS) and the Lonestar system at the Texas Advanced Computing Center (TACC). The Jaguar Cray XK6 system has 18,688 nodes connected through a Gemini internal interconnect, and each node has a single 16-core AMD 6200 series Opeteron processor. The Lonestar has 1,888 compute nodes, each of which contains two hex-core Intel Xeon processors, 24GB of memory and a QDR InfiniBand switch fabric that interconnects the nodes through a fat-tree topology. The system supports a 1PB Lustre parallel file system.

Multiple experiments using synthetic application codes were performed on Jaguar Cray XK6 system, in order to evaluate and analyze the framework performance and scalability. We also performed experiments with a coherent turbulent vortex application with feature-based object extracting on Lonester system, in order to test system behavior in real scientific scenario.

A. System Performance and Scalability

1) Weak Scaling Experiments: This experiment measures and evaluates the performance of our system for a weak scaling scenario. In this experiment, two synthetic application codes are used to insert various size of data into system server side, register data events for data region of interest, get notified and retrieve data blocks if all the data exist. These two codes emulate a simple and generic data publishing and event subscribing workflow in a coupled interactive pattern in real scientific scenarios. The publisher application and subscriber application are distributed and discretized running on different computing resources, but sharing the same global communication domain and computation domain. The shared computation domain used in this case is a 3-dimensional Cartesian grid; and each application assigns distinct number of processors for each dimension, i.e., $X \times Y \times Z$. Similar to the real scientific scenario, the queried data can be mapped into the shared computation domain and be represented by using its geometric location information like *(x,y,z)*. The data size in a certain region can be easily acquired by *data point size* $\times$ *region volume*.

Table I shows the test setups of our experiments. The number of data publisher processors and consequent number of concurrent data inserting varies from 64 to 8192, each of which inserts 32MB data. Accordingly, the number of subscriber processors and corresponding concurrent event subscriptions and data retrieving is from 8 to 1024. Each subscriber will retrieve 128MB data. The number of server nodes on staging area is from 2 to 256, which means every node needs to offer 1GB memory for data storage. The processor distribution of each application and queried data size are also presented.

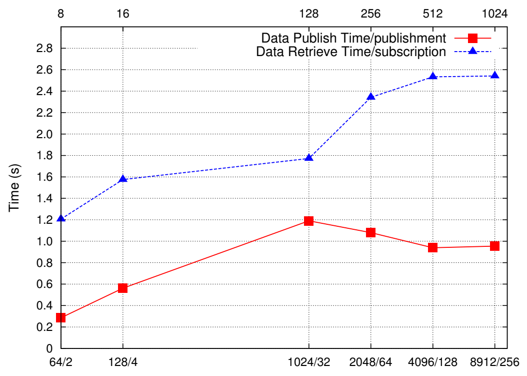


Fig. 5. Execution time for data publishing and data retrieve. Bottom X axis: number of publishers / number of staging servers; Top X axis: number of subscribers.

On the publisher side, we measured the execution time including the total time to prepare data and transfer data to the staging area at the application level; on the subscriber side, since the subscriber operated in a non-blocking manner, we separately measured the time spent on subscription submission, notification receiving and data retrieving. The final

Number of Publishers	64	128	1024	2048	4096	8192
Number of Subscribers	8	16	128	256	512	1024
Number of Staging Nodes	2	4	32	64	128	256
Query Data Size	2GB	4GB	32GB	64GB	128GB	256GB
Processor Distribution on Publisher side	$4 \times 4 \times 4$	$8 \times 4 \times 4$	$16 \times 8 \times 8$	$16 \times 16 \times 8$	$16 \times 32 \times 8$	$16 \times 32 \times 16$
Processor Distribution on Subscriber side	$2 \times 1 \times 4$	$4 \times 1 \times 4$	$8 \times 2 \times 8$	$8 \times 4 \times 8$	$8 \times 8 \times 8$	$8 \times 8 \times 16$

TABLE I
EXPERIMENT SETUP FOR SYSTEM PERFORMANCE AND SCALABILITY EVALUATION.

timing value averages the execution time in 100 iterations over the total number of processor for each application. The subscription submission time is constantly around 0.00016 seconds for all the experiment sets, which shows very low overhead and light impact on local computation. Figure 5 illustrates the execution time for data publishing and data retrieving, presenting a nice overall scalability with the various data size and number of application processors for concurrent region based data query and event registration. In terms of data publishing, the time difference between the largest scale case and the smallest one is only around 1.1 seconds; and that value is less than 1.5 seconds on the subscriber side. However, the trend of increasing time of data publishing and retrieval can be identified on the plot with larger application scale. The reason is, as the number of the staging servers increases, the storage of data is distributed over more servers, which causes the longer coordination across the staging sever side and introduces more message passing between servers and clients. Also, the interference on the shared links causes higher delays when using 32 and 64 staging servers for data publishing.

Figure 6 illustrates the application level data exchange throughput by using our system. The point values are calculated based on the execution time in Figure 5 and the exchanged data size. The application level throughput increases dramatically when the system scales up.

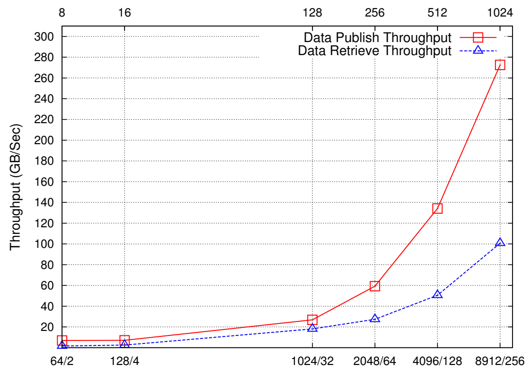


Fig. 6. Application level throughput for data publishing and data retrieve. Bottom X axis: number of publishers / number of staging servers; Top X axis: number of subscribers.

2) *Events Scaling Experiments:* In this part, the experimental setups and testing sets are the same as the previous experiments, except for the data event registered from the subscriber side. Instead of subscribing the data block in the entire data domain, we subscribe the minimum value in the whole computation domain. The subscription could be described as followed: if the minimum data value in a region of interest is greater than a threshold, notify the clients with the coordinates of corresponding data element in minimum value. The events number varies with the change of subscriber number, from 8 to 1024, and each query targets an application data domain of 16,777,216 data elements (128MB/8Byte) distributed stored on the server side.

Although the client immediately returns after submitting the data event and won't communicate with server side before getting notified, the event processing capability at server side is still important and may impact the overall application. We measure the system response time from the server side, which includes the time spent on subscription management, data monitoring, distributed computation and result aggregation.

Figure 7 presents the system response time and event processing rate over different experimental sets. As we can see, the system response time increases a little bit with the scaling up of application processors and query numbers. The time difference between the shortest response time and longest one is around 2 seconds, which is greater than time variation on data retrieving in the previous experiment. There are three main reasons for this gap: (1) the data of interest is distributed across multiple servers. As a result, the divide-and-conquer method with the binary aggregation tree introduces the communication overhead; (2) each leaf server in the binary aggregation tree needs to spend time on computation; (3) imbalanced data storage distribution may also impact the processing time. However, it's worth pointing out that in our messaging system, the event matching and result aggregation at server side are fully asynchronous and run in parallel with workflows at client side, therefore this 2 second time difference is acceptable for parallel information query of large data. In addition, the event processing rate under concurrent subscribing dramatically increases with the increasing subscription number.

3) *Data Scaling Experiments:* In this experiment, 64GB of data is inserted into 64 staging nodes from 2048 publishers, and 256 subscribers concurrently submit the data query events about various data region of interest, which contain data from 4GB to 64GB (each subscriber queries 16MB to 256MB data). The subscription used in this case contains both the minimum value monitoring and data retrieving: if the minimum data value in the region of interest is less than a threshold, the subscriber will get notified and then fetch all the data inside

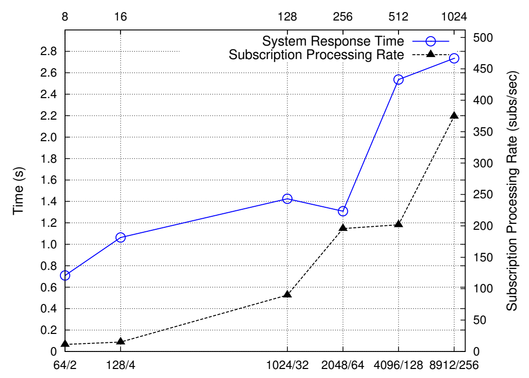


Fig. 7. System response time and event processing rate for events scaling experiment. Bottom X axis: number of publishers / number of staging servers; Top X axis: number of subscribers. Left Y axis: time; Right Y axis: event processing number per second

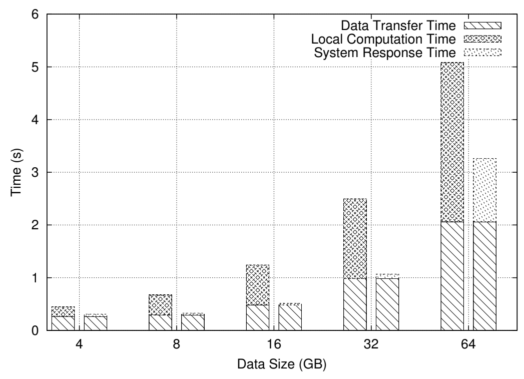


Fig. 9. Comparison with traditional post-processing approach. Left column: performance using traditional post-processing approach; right column: performance using our system.

this region. System response time and data retrieving time are measured and analyzed in this experiment.

As shown in Figure 8, system response time shows very good scaling performance with 0.06 seconds difference when the registered data size changes from 4GB to 32GB. However, when subscribing the entire 64GB data distributed on the server side, the system response time jumps to 1.2 seconds. The most significant reason is, based on the Hilbert SFC DHT, the smaller region of data subscribing has higher possibility to be hashed into fewer servers for data storage location information lookup and have fewer storage servers involved, which reduces the depth of aggregation binary tree and consequently the communication overheads during the result aggregation. Furthermore, the data transfer time increases with the change of the data size, under a constant number of subscriber processors.

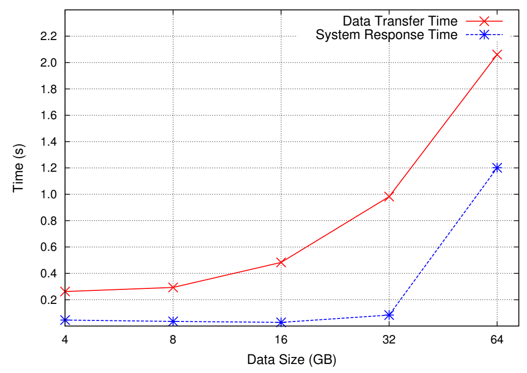


Fig. 8. System response time and data transfer time with data scaling from 4GB to 64GB.

B. Comparison with Traditional Approach

One of the objective of our system is to avoid the overheads of continuously monitoring data-availability and retrieving the data to detect the events of interest and of unnecessarily moving data. To validate this advantage of our system, we repeat the previous test case by using a traditional approach that offloads the entire data block into subscriber nodes first and then performs local post-processing. Overall performance of these two approaches are compared.

The left bar in Figure 9 present the result from experiment with traditional approach in various data size, which includes the entire data transfer time (lower part) and local computation time (upper part). The right bar represents the system response time plus data transport time by deploying our system. In terms of data computation and overall time cost, the result shows an overwhelming advantage with our approach by offloading the data computation and monitoring to staging area through associated data event subscription. It's also worth noting that, the non-blocking subscription mechanism ensures the client side continues the computation while the system processing on staging nodes is running. In addition, if the minimum data value doesn't meet the constraint conditions that the subscribers set, the data transfer in the traditional case becomes unnecessary.

C. Effectiveness in Coherent Turbulent Vortex Scenario

This section presents the behaviors and effectiveness of our messaging system running with a coupled coherent turbulent vortex [18] application with the requirement of in-situ feature-based object visualization on Lonestar system. The scientific dataset is generated on the publisher side by simulation of coherent turbulent vortex structures with 128^3 resolution (vorticity magnitude) at 200 time steps. The data event subscriptions with feature information that defines the data objects of interest are submitted from the clients to detect the associated data objects in a certain data domain, which is defined as a thresholded connected voxel region evolving both in location and shape during the simulation.

In this experiment case, we define the volume regions of interest and the vorticity values both in the range of 9 to maximum as the registered feature. A feature detecting algorithm implemented on the system server side keeps tracking the data object satisfying the registered feature. If the featured object is tracked inside the *data point region*, the server side will notify the clients and trigger the in-transit visualization

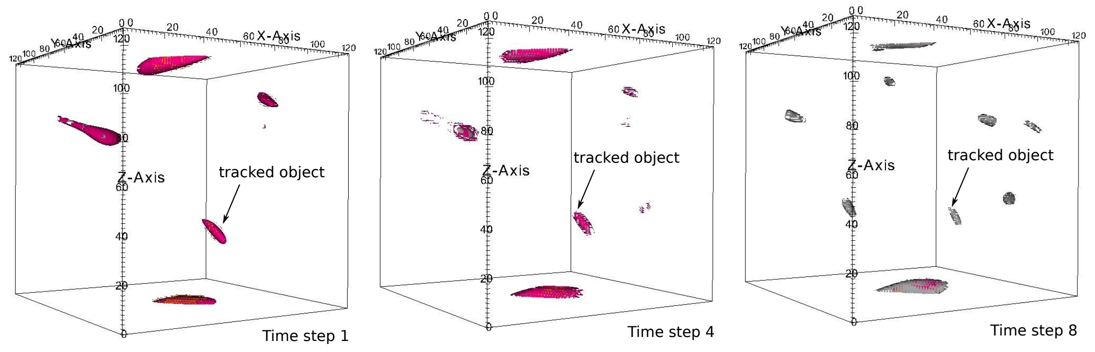


Fig. 10. Sample view of the evolving volume regions with featured objects visualized on staging nodes when registered event (feature is detected in this region) occurs.

code. Sample results at *(Time Step 1, Time Step 4, and Time Step 8)* are shown in Figure 10. We can clearly find that, only the data object that satisfies the feature constraint in the region of interest is tracked and visualized on staging area. It avoids the overhead spent on unnecessary data movement and visualization.

VI. Related Work

Insitu/Intransit Processing and Data Staging: The increasing performance gap between the compute and I/O capabilities has motivated recent developments in both *insitu* and *intransit* data processing paradigms. Largely data-parallel operations, including visualization [19]–[21], and statistical compression and queries [22], have been directly integrated into the simulation routines enabling them to operate on in-memory simulation data. Another approach, used by FP [23] and CoDS [24], performs insitu data operations on-chip using separate dedicated processor cores on multi/many-core nodes.

Data staging area, namely, a set of additional compute nodes allocated by users when launching the parallel simulations, has been investigated in projects such as DataStager [3], PreDatA [4], JITStaging [25], DataSpaces [1]/ActiveSpaces [2], and Glean [5]. Most of these existing data staging solutions primarily utilize the data transport mechanism of underlying hardware and memory resources in staging area to quickly and asynchronously move the data off simulation nodes to lessen the impact of expensive I/O operations. They typically support limited one-step synchronous data operations within the staging area, such as preprocessing, and transformations, often resulting in underutilization of the computational power of the staging nodes. Our approach implements a notification-based messaging system, allowing users to register the data events of interests and consequent actions to staging area in a non-blocking manner and effectively leveraging the memory resources and computation capabilities of the staging area.

Publish/Subscribe based System in HPC: Since the publish/subscribe messaging pattern provides asynchronous and flexible event registration, it is well suited for many interactive and coordinated large-scale applications. An existing event-based system [7] has investigated this nature to offer decoupled communication to aid system scalability and adaptability. Cayuga [8] extends the expressiveness of standard publish/subscribe system to powerful language features. Some researchers from UK even propose a common API [9] of publish/subscribe systems for research requirements.

However, only few projects apply this pattern into HPC applications. The Echo [10] publish/subscribe system is a typical high performance event delivery middleware designed for large scale event rates in grid computing environment. For event filtering and transformation Echo uses channel-based filtering and extends event channels via derivation. All the required computation for filtering and transforming events are performed in the same source node for the original event channel. EVPath [11] is a next generation of high performance distributed event middleware in grid computing environment, which is an extended and upgraded version of Echo [10] with system resource monitoring events. Another publish/subscribe based event service for HPC applications is designed and implemented with CCA [26], in order to achieve performance enhancement.

Compared with these systems, Our scalable framework extends the capability of standard publish/subscribe pattern and provides more flexible and complex content-based real time data monitoring as well as user-specific insitu/intransit analytics for specific data intensive HPC scenarios.

VII. Conclusion and Future Work

In this paper, we present the architecture, design and implementation of a scalable notification-based messaging system that builds on the data staging resources in HPC environment as well as the customized in-situ/in-transit analytics defined there, and provides a publish/subscribe messaging pattern to scientists. The system allows the scientists to dynamically register the live data events of interest, define the actions that are triggered based on the events, and get notified when the events occur. In addition, our system is implemented and deployed on the leading HEC platforms, such as Jaguar Cray XK6 machines at Oak Ridge National Laboratory and the Lonestar system at the Texas Advanced Computing Center (TACC). Finally, the experimental results and evaluation validate the performance advantages of our system, and the potential for real scientific applications. Overall, our system is extensible to a wide range of large scale scientific application scenarios and can potentially accelerate the scientific discovery and improve productivity.

Our direction for future work includes dynamic code integration on staging servers of our system by using scripting language and measure the related performance. In addition, we are also trying to integrating our system with very large data intensive applications for scientific debugging.

ACKNOWLEDGMENT

The research presented in this work is supported in part by National Science Foundation (NSF) via grants numbers IIP 0758566 and DMS-0835436, by the Department of Energy ExaCT Combustion Co-Design Center via subcontract number 4000110839 from UT Battelle, by the Scalable Data Management, Analysis, and Visualization (SDAV) Institute via the grant numbers DE-SC0007455 and via grant number DE-FG02-06ER54857, and by an IBM Faculty Award. The research and was conducted as part of the NSF Cloud and Autonomic Computing (CAC) Center at Rutgers University.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf, "PreDatA - preparatory data analytics on peta-scale machines," in *Proc. of 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS'10)*, April 2010.

[5] 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*, oct. 2011, pp. 9 –14.

[6] N. Jiang, A. Quiroz, C. Schmidt, and M. Parashar, "Meteor: a middleware infrastructure for content-based decoupled interactions in pervasive grid environments," *Concurr. Comput. : Pract. Exper.*, vol. 20, no. 12, pp. 1455–1484, Aug. 2008. [Online]. Available: http://dx.doi.org/10.1002/cpe.v20:12

[7] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," 2003.

[8] A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White, "Towards expressive publish/subscribe systems," in *In Proc. EDBT*, 2006, pp. 627–644.

[9] P. Pietzuch, D. Eyers, S. Kounev, and B. Shand, "Towards a common api for publish/subscribe," in *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, ser. DEBS '07. New York, NY, USA: ACM, 2007, pp. 152–157.

[10] G. Eisenhauer, K. Schwan, and F. Bustamante, "Publish-subscribe for high-performance computing," *IEEE Internet Computing*, vol. 10, no. 1, pp. 40–47, Jan. 2006.

[11] G. Eisenhauer, M. Wolf, H. Abbasi, and K. Schwan, "Event-based systems: opportunities and challenges at exascale," in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '09. New York, NY, USA: ACM, 2009, pp. 2:1–2:10.

[12] M. Day, J. Bell, P.-T. Bremer, V. Pascucci, V. Beckner, and M. Lijewski, "Turbulence effects on cellular burning structures in lean premixed hydrogen flames," *Combustion and Flame*, vol. 156, pp. 1035–1045, 2009.

[13] T. Echekki and J. H. Chen, "Direct numerical simulation of autoignition in non-homogeneous hydrogen-air mixtures," *Combust. Flame*, 2003, to appear.

[14] A. Mascarenhas, R. W. Grout, P.-T. Bremer, E. R. Hawkes, V. Pascucci, and J. Chen, *Topological feature extraction for comparison of terascale combustion simulation data*, ser. Mathematics and Visualization. Springer, 2011, pp. 229–240.

[15] J. Bennett, V. Krishnamoorthy, S. Liu, R. Grout, E. Hawkes, J. Chen, J. Shepherd, V. Pascucci, and P.-T. Bremer, "Feature-based statistical analysis of combustion simulation data," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 12, pp. 1822 –1831, dec. 2011.

[16] C. Docan, M. Parashar, and S. Klasky, "Dart: a substrate for high speed asynchronous data io," in *Proc. 17th International Symposium on High Performance Distributed Computing (HPDC'08)*, 2008.

[17] R. Alverson, D. Roweth, and L. Kaplan, "The gemini system interconnect," in *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*, aug. 2010, pp. 83 –87.

[18] J. Chen, D. Silver, and M. Parashar, "Real time feature extraction and tracking in a computational steering environment," in *Proceedings of the High Performance Computing Symposium, HPC2003, Society for Modeling and Simulation International*, 2003, p. 155160.

[19] H. Yu, C. Wang, R. Grout, J. Chen, and K.-L. Ma, "In Situ Visualization for Large-Scale Combustion Simulations," *IEEE Computer Graphics and Applications*, vol. 30, no. 3, pp. 45–57, 2010.

[20] J.-M. F. Brad Whitlock and J. S. Meredith, "Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System," in *Proc. of 11th Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'11)*, April 2011.

[21] N. Fabian, K. Moreland, D. Thompson, A. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K. Jansen, "The paraview coprocessing library: A scalable, general purpose in situ visualization library," in *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, oct. 2011, pp. 89 –96.

[22] S. Lakshminarasimhan, J. Jenkins, I. Arkatkar, Z. Gong, H. Kolla, S.-H. Ku, S. Ethier, J. Chen, C. Chang, S. Klasky, R. Latham, R. Ross, and N. Samatova, "Isabela-qa: Query-driven analytics with isabela-compressed extreme-scale scientific data," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, nov. 2011, pp. 1 –11.

[23] M. Li, S. S. Vazhkudai, A. R. Butt, F. Meng, X. Ma, Y. Kim, C. Engelmann, and G. Shipman, "Functional partitioning to optimize end-to-end performance on many-core architectures," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10, Washington, DC, USA, 2010, pp. 1–12.

[24] 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 (IPDPS'12)*, 2012.

[25] 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)*, June 2011.

[26] I. Gorton, D. Chavarria-Miranda, M. Krishnan, and J. Nieplocha, "A high-performance event service for hpc applications," in *Software Engineering for High Performance Computing Applications, 2007. SE-HPC '07. Third International Workshop on*, may 2007, p. 1.