Cluster Comput (2015) 18:29–40
DOI 10.1007/s10586-014-0396-6

In-situ feature-based objects tracking for data-intensive scientific and enterprise analytics workflows

Solomon Lasluisa · Fan Zhang · Tong Jin · Ivan Rodero · Hoang Bui · Manish Parashar

Received: 23 January 2013 / Revised: 29 September 2013 / Accepted: 17 July 2014 / Published online: 22 August 2014

Abstract Emerging scientific simulations on leadership class systems are generating huge amounts of data and processing this data in an efficient and timely manner is critical for generating insights from the simulations. However, the increasing gap between computation and disk I/O speeds makes traditional data analytics pipelines based on post-processing cost prohibitive and often infeasible. In this paper, we investigate an alternate approach that aims to bring the analytics closer to the data using in-situ execution of data analysis operations. Specifically, we present the design, implementation and evaluation of a framework that can support in-situ feature-based objects tracking on distributed scientific datasets. Central to this framework is a scalable decentralized and online clustering, a cluster tracking algorithm, which executes in-situ (on different cores) in parallel with the simulation processes, and retrieves data from the simulations directly via on-chip shared memory. The results from our experimental evaluation demonstrate that the in-situ approach significantly reduces the cost of data movement, that the presented framework can support scalable feature-based objects tracking, and that it can be effectively used for in-situ analytics in large scale simulations.

Keywords Simulations workflows · Scientific data analysis · Scalable in-situ data analytics · Feature-based objects tracking

S. lasluisa · F. Zhang · T. Jin · I. Rodero (✉) · H. Bui · M. Parashar
Rutgers Discovery Informatics Institute, NSF Cloud and Autonomic Computing Center, Rutgers University, Piscataway, NJ, USA
e-mail: irodero@cac.rutgers.edu

S. Lasluisa
e-mail: lasluisa@cac.rutgers.edu

F. Zhang
e-mail: zhangfan@cac.rutgers.edu

T. Jin
e-mail: tjin@cac.rutgers.edu

H. Bui
e-mail: hbui@cac.rutgers.edu

M. Parashar
e-mail: parashar@cac.rutgers.edu

1 Introduction

Extreme scale systems enable scientists as well as enterprises to run large application workflows that produce and/or process large amounts of data. For example, scientists and engineers are able to conduct simulation studies of natural and engineering phenomenon with unprecedented accuracy using detailed models and exploring large parameter spaces. Similarly, enterprises are able to explore networks of human and system interconnectivity and interactions and use the resulting insights to identify trends and inform business strategy. The ability to analyze and visualize data in a timely and scalable manner is critical to these applications.

However, growing system and application scales are also resulting in unprecedented data volumes and data rates, and the complexity and costs (both, latency and energy) associated with managing, transporting and processing (e.g., analyzing, visualizing, archiving, etc.) this data have become significant challenges. unprecedented amount of data. To enable scientific discovery, the high amount of simulation data has to be analyzed and understood by domain scientists. However, The increasing gap between computation and disk I/O speeds is making traditional data management and data analytics pipelines based on post-processing cost prohibitive and often infeasible [1]. For example, moving entire datasets from a large scale system running a simulation to remote storage and analytics servers is becoming prohibitively expensive in terms of the the time required as well as its energy costs [2]. Similarly, the efficiency and scalability of subsequent analysis operations are also hindered by the costs of disk-based I/O. These challenges are quickly limiting the ability to effectively transform data into insights, and require rethinking traditional data analytics pipelines to reduce data movement. Recently, in-situ and in-transit data processing pipelines have emerged as a promising approach [3] to effectively reduce overheads [4] and energy costs [5] due to data movement by placing data processing operations closer to where the data is being produced.

In this paper, we use this approach to enable scalable in-situ feature-based object tracking for large scale scientific simulations. In order to extract insightful information from large datasets produced by simulations over thousands of time steps, scientists often need to follow data objects of interest (i.e., features) across the different time steps. For example, meteorologists track storm formation and movement in climate modeling simulation while physicists identify burning regions in combustion simulations. As a result, feature extraction and tracking is an important technique for analyzing and visualizing scientific datasets. However, most feature extraction and tracking techniques operate offline by post-processing data written into files by the simulation runs. Being able to perform such feature-based analytics in-situ, i.e., concurrently with a simulation itself, can significantly improve the utility of these techniques at large scale. It can also lead to better utilization of expensive high-end resources as well as the overall productivity of the simulations

However, performing in-situ feature tracking presents several research challenges. First, it requires a distributed feature extraction and tracking algorithm that operates on distributed data. Second, it requires a programming and runtime system that enables the mapping and execution of the simulation and the data analysis operations on co-located processor cores, and supports asynchronously data sharing at runtime. Most existing in-situ data analysis implementations employ an inline approach, i.e., the data analysis operations are embedded within the execution path of the main simulation process, usually as function calls. A major drawback of this approach is that the simulation has to block and wait for the completion of the in-situ analytics routine, which impacts the execution and performance of the main simulations. This approach also requires significant modification of the simulation code, which is undesirable.

In this paper, we present the design and implementation of a framework that can support in-situ feature-based objects tracking for large-scale parallel simulations. Central to this framework is the scalable decentralized and online clustering (DOC) [6,7] and cluster tracking algorithm, which executes in-situ, i.e., on different cores, and in parallel with the simulation processes, and accesses simulation data directly and asynchronously (to the extent possible) via on-chip shared memory. The framework also provides programming support for composing in-situ "simulation plus analytics" workflows. Results from an experimental evaluation of the framework on the Lonestar system at Texas Advanced Computing Center (TACC) are also presented and demonstrate that the in-situ approach significantly reduces the cost of data movement, that the presented framework can support scalable feature-based cluster tracking, and that it can be effectively used for in-situ analytics for large scale simulations.

The rest of the paper is structured as follows. Section 2 provides background on data-intensive analytics workflows. Section 3 presents the motivating use cases. Section 4 describes the overall system architecture and the implementation of the feature-based tracking algorithm. Section 5 presents the experimental evaluation of the prototype system using s 3D time-varying CFD dataset. Section 6 discusses related work. Section 7 concludes the paper and outlines directions for future work.

2 Background

2.1 Data-intensive analytics workflows

This research targets data-intensive analytics workflows that are becoming increasingly important for both large scale scientific and enterprise data analysis, and specifically focuses on scientific simulation workflows.

Traditionally, large scale data analysis are performed offline as a post-processing step. For example, scientific simulations write data to the file system, which is then read by the analysis or visualization codes. However, given the increasing scale of the applications and the costs associated with I/O, end-to-end data-intensive analytics workflows that integrate applications with online analysis operations are more attractive. These workflows are composed of component applications that frequently share or exchange large volumes of data at runtime. Figure 1 shows the data analytics workflow for a turbulent combustion simulation S3D [8], where three types of analytics operations, i.e., descriptive statistics, visualization and topological analysis are being performed on the simulation data on the fly.

The couplings,interactions and coordination associated with data-intensive scientific workflows can be expressed using three canonical workflow patterns [9,10] as below:

- **Tight coupling**: In this class of workflows, the coupled component applications run concurrently and exchange data frequently, as in the case of online feature-based objects tracking. Due to the large volume of data, data movement costs can quickly dominate the execution times in these workflows.

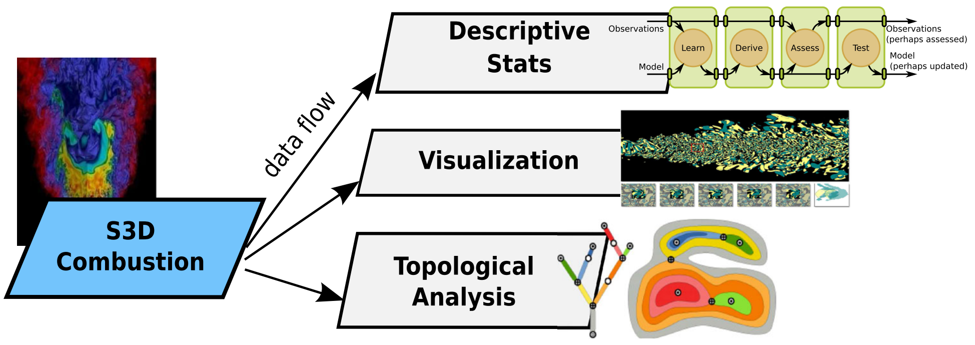


Fig. 1 Data analytics workflow for the S3D combustion simulation [4]

- **Loose coupling**: In this class of workflows, coupling and data exchanges are less frequent, often asynchronous and possibly opportunistic. The coupled component applications in this case may run concurrently or sequentially.
- **Dataflow coupling**: This class represents more typical workflows often expressed as a dataflow-based directed acyclic graph (DAG) where data produced by one application flows through one or more data processing pipelines. The S3D combustion pipeline is an example of this pattern.

This paper focuses on the tight coupling pattern exhibited by the online feature-based objects tracking analysis workflow.

2.2 In-situ execution of data analytics workflows

It is clear from the discussion above that reducing the overheads of data movement during the interactions in a tightly coupled component application workflow can significantly improve performance and scalability. However, existing frameworks typically run different applications of a tightly-coupled data-intensive workflow on separate sets of compute nodes, which can result in a large amount of network data movement.

One attractive approach for reducing data movement is to perform runtime data analytics in-situ, directly on the compute node where data is generated, so as to maximize intra-node data sharing. This is illustrated in the example in Fig. 2 where the simulation and analysis components of the end-to-end workflow run concurrently on the same compute node, but on different processor cores. In that case, intra-node data transfers between the coupled applications can be performed using shared memory.

3 Motivating use cases

3.1 Scientific scenario

To extract scientific insights from data generated by time-varying simulation, scientists need to study the evolution of different physical phenomena by tracking objects of interest over time. For example, in 3D computational fluid dynamics simulations that contain evolving amorphous regions, connected voxel regions of interest—features—needs to be identified at each time step and tracked over multiple time steps in order to visualize the time-varying datasets. Moreover, the traditional centralized and post-processing approach becomes infeasible due to overheads and cost associated with extracting, transporting and storing the large volume of data from compute nodes to storage servers. As a result, a distributed and online object tracking approach is critical to support data analysis and visualization for data-intensive scientific simulations.

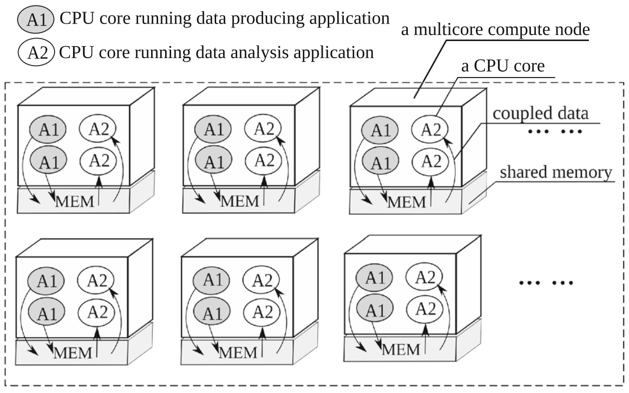

A1 CPU core running data producing application
A2 CPU core running data analysis application
a multicore compute node
a CPU core
coupled data
shared memory
A1
A2
MEM
… …

Fig. 2 In-situ execution of a data analytics workflow.

In this paper, we focus on analyzing the time-varying 3D dataset generated by simulation of coherent turbulent vortex structures. More specifically, we focus on tracking objects as thresholded connected voxel regions that evolve both in location and shape over time with in-situ execution of the "simulation plus analytics" workflow.

3.2 Enterprise scenario

Microblogging has become a very popular social media service that generates a huge number of messages daily in platforms such as Twitter. Twitter as a social microblogging platform allows users to post tweets (i.e., messages of 140 or less characters) that are made publicly available in realtime.

As Twitter gains popularity, its user-base grows and generates unprecedented amounts of data. As of September 2013, Twitter has over 200 million active users with a total of 400 million tweets sent per day. Microblogging social media, in general, becomes a valuable source of people's opinions and sentiments [11]. User-generated data can be effectively used for business, social and political studies. However, the huge data volume poses great challenges, such as:

- Large scale: millions of users worldwide;
- Rich content: high variety (or types) of data generated by users;
- Complex interaction: constant interactions between users and content; and
- Dynamic nature: content is constantly being updated and evolves over time.

Our approach can be applied to social media data analysis at a large scale. More specifically, it can be used to detect patterns, to mine trends and to analyze sentiments from tweet streams. Real-life scenarios include predicting election outcome, tracking storm movement and detecting flu outbreak. We could answer these questions by mining tweet streams at a large scale. Some of the associated tasks are:

- Define an event, such as healthy, disease, election.
- Sentiment learning, such as negative, neutral, or positive.
- Identify how an event spreads and evolves.
- Model event spread such as information propagation.
- Visualization.

4 Design and implementation

4.1 Overview of the system framework

The system architecture of the proposed distributed data analysis framework consists of three main components: execution clients, in-situ data staging daemons and a management server. Each execution client abstracts and represents a basic computation element such as a physical processor core. The management server acts as the rendezvous point to bootstrap execution clients, and manages the execution of various applications within the in-situ data analysis workflow.

Figure 3 shows co-located execution of DOC workers with the simulation program, where the physical processor cores on the multi-core compute nodes are functionally partitioned. As shown in the figure, on-node computation resource is mostly used by the scientific simulation program, and two processor cores are used to execute in-

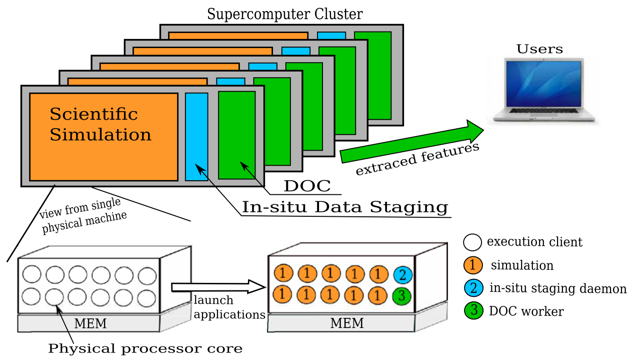


Fig. 3 Architecture of the in-situ feature extraction and tracking system

situ data staging daemon and DOC worker. The in-situ data staging daemons across the distributed cluster nodes build a Co-located DataSpaces (CoDS) [12], which provides a virtual shared-space abstraction to support asynchronous and decoupled coordination and data sharing between simulation processes and the DOC worker. The capability of functional partitioning of node-level processor cores, and the programming model and runtime to support asynchronous communication between the intra-node cores, is critical to exploit the ever-increasing hardware parallelism on emerging supercomputers.

Our framework employs the data-centric scheduling approach to map processes from different applications onto physical processor cores so that a large portion of the inter-application data transfers can be performed using the on-node shared memory. More specifically, two mapping methods - both centralized server side and decentralized execution client side - are designed in the framework. The server side mapping is applied to a "bundle" of concurrently coupled component applications that are launched simultaneously, have regular inter-application communication patterns and do not need dynamic re-mapping after launch, which matches with the "simulation plus DOC" scenario described in this paper. The execution client side approach schedules the newly launched workflow application that have dependency on the distributed data generated by preceding application. More details about the data-centric mapping is discussed in [12].

Each component application of our in-situ feature-based objects tracking system is assigned an unique id. Every execution client is associated with one specific component application (simulation or DOC) when the data-centric mapping is completed. Then each execution client is colored with the value of the assigned id, and execution clients with the same color would form a processes group and collectively launch the associated application.

Because scientific simulation and the DOC worker run as separate programs on different physical cores, data structures cannot be directly shared under single process address space as of the inline approach. As a result, efficient intra-node data communication is crucial to the performance. Our framework utilizes the intra-node shared memory for passing data between simulation processes and DOC workers. To avoid the overhead of user-space double-copy, the framework implements both single copy and zero copy schemes. In the single copy optimization, simulation data is copied from user space to the shared memory allocated by in-situ data staging daemon, and then delivered to the receiver, e.g., DOC worker, without copying. In the zero copy optimization, simulation processes can pre-allocate buffer from the shared memory by sending allocation request to in-situ data staging daemon, then writes data directly into the shared memory buffer without copying.

4.2 Feature-based objects tracking using DOC

4.2.1 Decentralized online clustering (DOC)

DOC, introduced by Quiroz et al. [6,13], was created to provide online and decentralized data analysis, using the collective computing resources in distributed systems. DOC is an online algorithm, thereby allowing short-term system behavior to be captured, as opposed to an offline approach that cannot capture short-term behavior. DOC has also been tested for its accuracy of identifying clusters within a given dataset and shown to perform as accurate as its offline counterparts such as *k*-means.

The approach of DOC is to divide the data space into regions and to assign a region to each processing node. DOC's underlying message substrate takes care of routing data points to the nodes responsible for their regions, with bounded costs in terms of the number of messages and the number of nodes involved [14]. Each node can then detect the number of points within each region. If the total number of points in the information space is known, then a baseline density for a uniform distribution of points can be calculated and used to estimate an expected number of points per region. Clusters are recognized within a region if the region has a relatively larger point count than this expected value. Conversely, if the point count is smaller than expected, then these points are potential outliers. However, clusters may cross region boundaries, and this must be taken into account when verifying potential outliers. The process described effectively determines the similarity between data points by the density of points within regions of the information space.

4.2.2 Features extraction

A method for identifying objects in scientific simulations is through data point clustering. Clusters can represent regions of interest in a dataset, for example a flame or region of chemical reaction in a combustion simulation. The distributed nature of data generation in scientific applications, as well as other applications, requires specialized clustering algorithms such as DOC. DOC's decentralized programming can reduce overall data transfer cost by its novel clustering workflow, where individual nodes are assigned a specified region of space to analyze. DOC workers run on each computational node and identify local clusters. These workers then communicate with neighboring workers and merge clusters iteratively. DOC can maintain data locality in certain configuration and thereby remove the need to aggregate data.

In order to identify clusters across time, we define the concept of cluster features. For any given cluster found within a dataset at some point in time, clusters features are a number of derived metrics and/or metadata that can be extracted from them. Examples of cluster features are location, density, bounds, size, and cluster metadata (e.g., points' origin).

The key to cluster identification is to determine the similarity between a set of cluster features. If two or more sets of cluster features are sufficiently similar over time, then they can be considered to correspond to the same cluster. Meta-clustering is applied to identify similarity by clustering the features extracted from the data. Just as with the original points, we take advantage of clustering technique for its ability to identify similarity without necessarily having to determine an absolute threshold beforehand.

Figure 4 shows an example of how a cluster feature can be used to track a cluster's movement. In this figure, three clusters found at different points in time are shown with their corresponding centroids. As can be seen, the three centroids form a cluster when seen as points in the same space. Based on this information, we can say that it is likely that the three clusters represent the same element or behavior pattern over time, and that, consequently, can likely be identified as the same cluster.

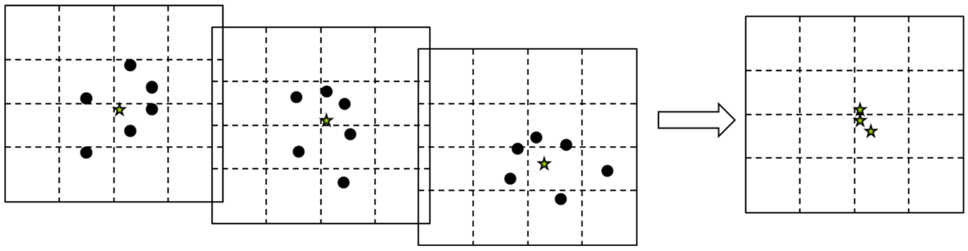


Fig. 4 Example of how cluster centroids (a cluster feature) can be used to track a cluster's movement [13]

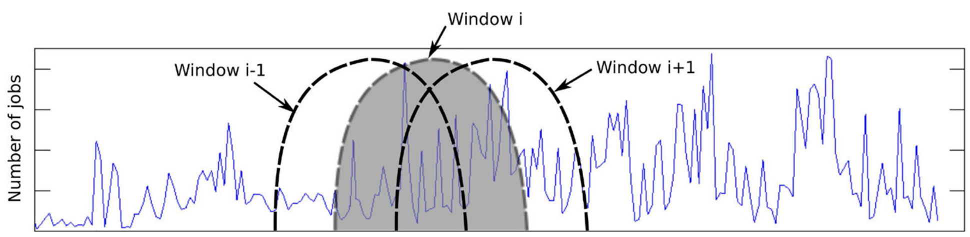


Fig. 5 Sliding window technique example

4.2.3 Cluster tracking with a sliding time window

Despite the possibility of observing stable sets of features over time, some features will necessarily keep changing because of the dynamics of the system. The purpose of cluster tracking is to be able to characterize and predict the behavior of the changing features whenever possible, along with the features that are static.

One possible approach to this is tracking static and dynamic features separately. However, determining which features to track in either way is not trivial. If only static features cluster over time, then using a single feature space for clustering will simply not work in either case. Our approach is then to define a sliding window for clustering that allows a fine-grained view of the features that change. More importantly, because changes are gradual, all features for a single identifiable cluster (not just those that are static) will form a cluster over some continuous time interval.

The sliding window technique captures incoming information from the data stream, saves a portion of previously captured information while discarding the oldest, and allows new information to replace the discarded information, as illustrated in Fig. 5. As can be seen in the figure, window$_i$ contains new requests from new time intervals, as well as some remaining requests from intervals contained in the previous window (window$_{i-1}$), but dropped the oldest time intervals that were contained in that window. The same rule is applied for subsequent windows.

Depending on the size of the time intervals, on the size of the step of the analysis window, and on the size of the analysis window itself, we can ensure that the changes in clusters are gradual from one analysis window to the next. Since the size as well as the step for each window can be changed, the rate of change between analysis windows can always be controlled. For our approach the default setting is to replace 10 % of the data points at each step. We arrived at this value empirically after finding that this level of accuracy was relatively high (when compared to other windows sizes ranging from 5 to 50 %) and maintained a low total computation cost.

Despite the gradual changes afforded by the sliding window technique, eventually the dynamics of the underlying data set will cause large changes in cluster features when compared across many analysis windows, as well as new clusters to appear, and other clusters to disappear. All of these cases reflect changes in system state caused by bothgradual and transitory events and conditions. Our feature tracking approach keeps track of the trajectories (paths) defined by feature sets across analysis windows, and is thus able to identify all of these changes. As a result, if patterns are identifiable in the paths, they can be used for prediction.

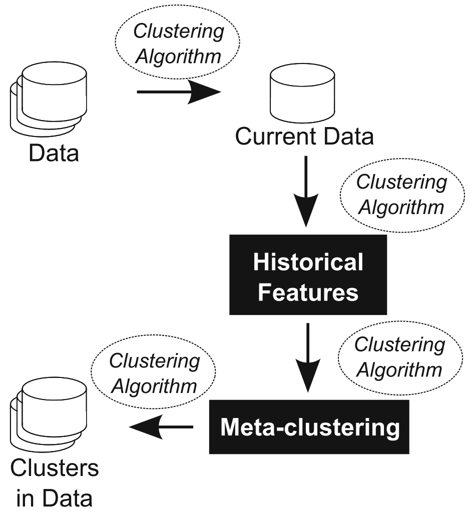


Fig. 6 Workflow for data cluster tracking

4.2.4 Cluster tracking algorithm

The following workflow, illustrated in Fig. 6, summarizes the six steps in the application of our algorithm.

(1) A data stream is read and, chronologically, an analysis window moves along the data, creating snapshots (windows) to analyze. The union of all windows makes the original data stream up to a point in time, but consecutive windows are not mutually exclusive (they overlap) as shown in Fig. 5.
(2) Each window is clustered to identify the relationships between data points within that specific window. Each of these first level clusters is a snapshot in time of a single cluster.
(3) For every window, each individual cluster has its features extracted. These cluster features will later be used to compare first level clusters from different windows to one another for cluster identification.

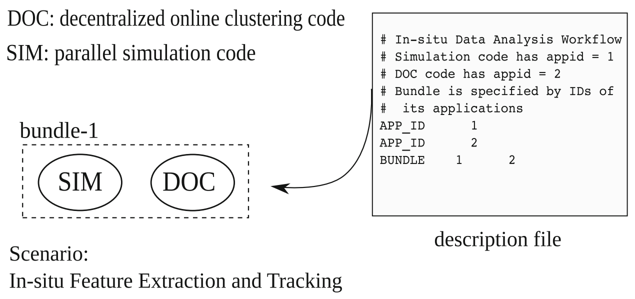
(4) Extracted cluster features are clustered with features from previous windows to identify how closely related they are to one another.
(5) The resulting meta-clusters can now be used to identify these related clusters as a single cluster to be tracked.
(6) The trajectory for each temporal cluster is updated by storing features in sorted order according to the analysis window in which they were found.

4.3 Composing in-situ data analysis workflows

One important idea of the proposed framework is to support building in-situ data analysis as a tightly-coupled workflow. Two problems need to be solved regarding the programmability: first, how to specify the control flow; second, how to program the coordination and data communication between the interacting workflow applications.

The tightly-coupled application workflow is expressed as a DAG, where each vertex in the DAG represents a parallel program. Our DAG representation extends traditional DAG representation such as DAGMan used in the workflow engine Pegasus, with the concept of a "bundle", which represents a group of parallel programs that need to be launched simultaneously, for example, the simulation and DOC programs of our in-situ feature extraction and tracking scenario. The edges of the DAG represent the control flow. The DAG as well as the bundles are explicitly defined by users. Figure 7 presents the DAG representation for our simulation and DOC application workflow, and the user-generated DAG description file which is then parsed by the management server. Note that each parallel program in the DAG is identified by a unique application id in the description file.

The components of the application workflow coordinate and communicate with each other through the abstraction of a shared data space-CoDS. In our in-situ feature extraction and tracking scenario, the DOC workers (data consumer) need to continuously access scientific data computed by the simulation (data producer). To implement this, each process of the simulation program specifies a descriptor as the key for its data, and inserts the data into CoDS using collective *put()* interface. Each DOC worker generates the query


DOC: decentralized online clustering code
SIM: parallel simulation code
bundle-1
SIM
DOC
description file
Scenario:
In-situ Feature Extraction and Tracking

```
# In-situ Data Analysis Workflow
# Simulation code has appid = 1
# DOC code has appid = 2
# Bundle is specified by IDs of
#  its applications
APP_ID      1
APP_ID      2
BUNDLE    1      2
```

Fig. 7 Example of workflow DAG representation and description file.

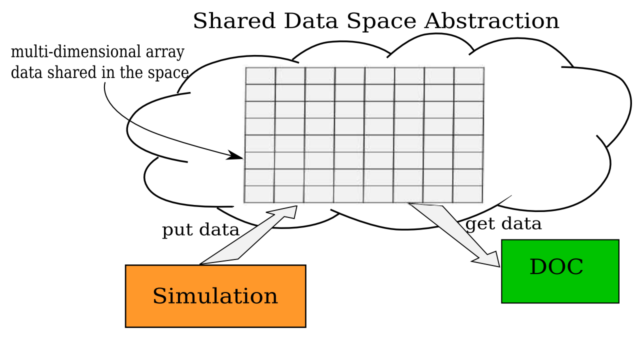

Shared Data Space Abstraction
multi-dimensional array data shared in the space
put data
get data
Simulation
DOC

Fig. 8 Simulation and DOC workers share data through data space abstraction.

key, and uses the *get()* interface to retrieve data of interest. More specifically, DOC workers use the interface *get_local()* to only retrieve simulation data produced on local machine. One assumption for our framework is that the interacting workflow applications share the knowledge about the data, such as type of key, data name and format. Figure 8 shows a logical view of data interaction through shared data space.

5 Experimental evaluation

The prototype implementation of our framework was evaluated on the Lonestar linux cluster at Texas Advanced Computing Center (TACC). The Lonestar has 1,888 compute nodes, and each compute node contains two hex-core Intel Xeon processors, 24 GB of memory and a QDR InfiniBand switch fabric that interconnects the nodes through a fat-tree topology. The system also supports a 1 PB Lustre parallel file system.

Our evaluation consists of two parts. The first part evaluates the end-to-end data transfer performance of our in-situ data analysis framework, and also compares it with the traditional disk I/O approach. The second part evaluates the effectiveness and accuracy of our DOC-based feature tracking algorithm, using a time-varying dataset generated by simulation of coherent turbulent vortex structures.

5.1 Performance of data transfer

This section evaluates the end-to-end data transfer performance, and more specifically the time used to transfer data from simulation processes to DOC workers, for both our in-situ memory-to-memory and the disk I/O approaches. In this case, we use a testing MPI program as the parallel data producing simulation, which runs on a set of m processor cores. The parallel DOC workers runs on a separate set of n processor cores where the ratio of m:n is 8. In our in-situ data analysis approach, each DOC worker runs on a processor core co-located with 8 simulation cores of the same

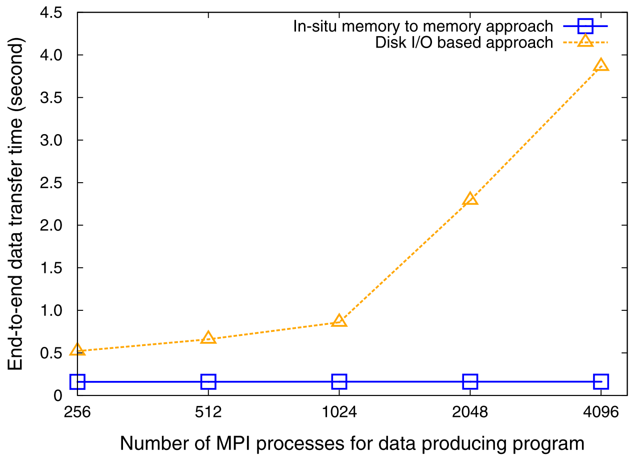


Fig. 9 End-to-end data transfer time in millisecond. The size of data produced per simulation process at each timestep is 32 MB

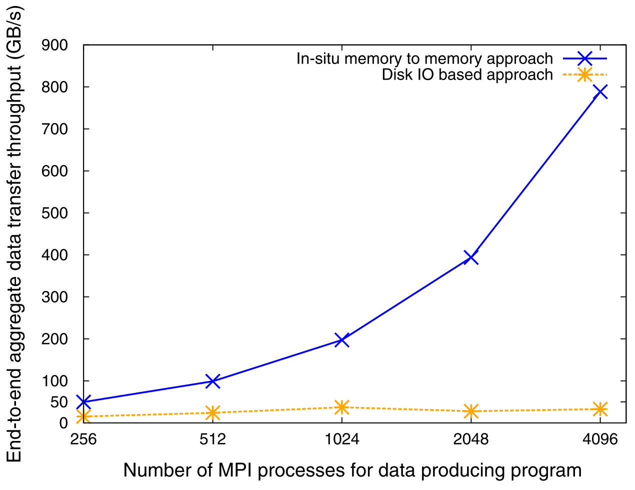


Fig. 10 Aggregate data transfer throughput. The size of data produced per simulation process at each timestep is 32 MB

compute node, and retrieves data generated by the 10 intra-node simulation processes through CoDS *get_local()* interface. In our framework, the *m:n* ratio can be configured by users. From our experience, the simulation part of the experiment is usually more compute intensive, thus it makes sense to allocate a large amount of cores to simulation tasks. On the other hand, the data analysis part (DOC in this case) is often data intensive hence requires a smaller number of cores. Our in-situ approach requires simulation and analysis tasks that exchange data to be scheduled in the same node in order to minimize data transfer and to reduce the overhead on the simulation itself. In the disk I/O approach, simulation processes dump data to disk with the one file per process method using binary POSIX I/O operations. Data files are then read by parallel DOC workers. For this evaluation, the number of simulation processes *m* is varied from 256 to 4,096, and the total data size produced by simulation at each timestep is varied from 8 to 128 GB. The testing program is configured to run for 100 timesteps at each data output size.

Figure 9 and 10 compare the performance of the two evaluated end-to-end data transfer approaches. As shown in Fig. 9, our in-situ memory-to-memory method is much faster than the disk I/O approach, with average speedup of transfer performance as about 10. Also, the in-situ memory-to-memory method is scalable, and shows no performance degradation when the number of MPI processes in data producing program increases from 256 to 4096. The reason for this significant performance gain is that all data movement is intra-node, and performed through the on-node fast I/O path-shared memory. But for the disk I/O approach, both data producer and DOC worker processes have to use the off-node slow path-disk. Figure 10 illustrates the performance gain from another dimension - aggregate data transfer throughput. The fast intra-node shared memory approach enables much higher aggregate bandwidth for the data movement between simulation and DOC.

5.2 Effectiveness of the feature-based cluster tracking algorithm

This section evaluates the effectiveness and accuracy of our proposed feature tracking algorithm, using a time-varying 3D dataset. The dataset is generated by simulation of coherent turbulent vortex structures with 128^3 resolution (vorticity magnitude) and 100 time steps. In this case, the data cluster or object of interest is defined as thresholded connected voxel regions. These regions evolve both in location and shape during the simulation. Although different time steps of the dataset can be visualized offline using visualization tools such as Visit, it is difficult to visually observe and accurately follow regions of interest. The tracking information from our algorithm is used to determine how the regions evolve, e.g., size, location, density, over the time steps.

In this experiment, we define the regions of interest as the data points with vorticity values in the range of 9 to maximum. Since the tracked objects in our experiment vary fast and last only around 10 time steps, we snapshot three time steps of the visualized dataset for the duration of the tracked object, as shown in Fig. 11. Also, it demonstrates the effective tracking of the evolving volume region (or object as in DOC) pointed by black arrows. We define *tracking accuracy* as the ratio of data points encompassed by the tracked objects to total number of points in the experiment. This ensures that high accuracy in object tracking can only be achieved by identifying paths from which the object pass through. In each experiment 50 frames were used to identify the paths of the objects within these 50 frames. The tracking accuracy across 47 tests was 92.28 % on average, meaning only 7.72 % of all vortex points were not associated to any trackable

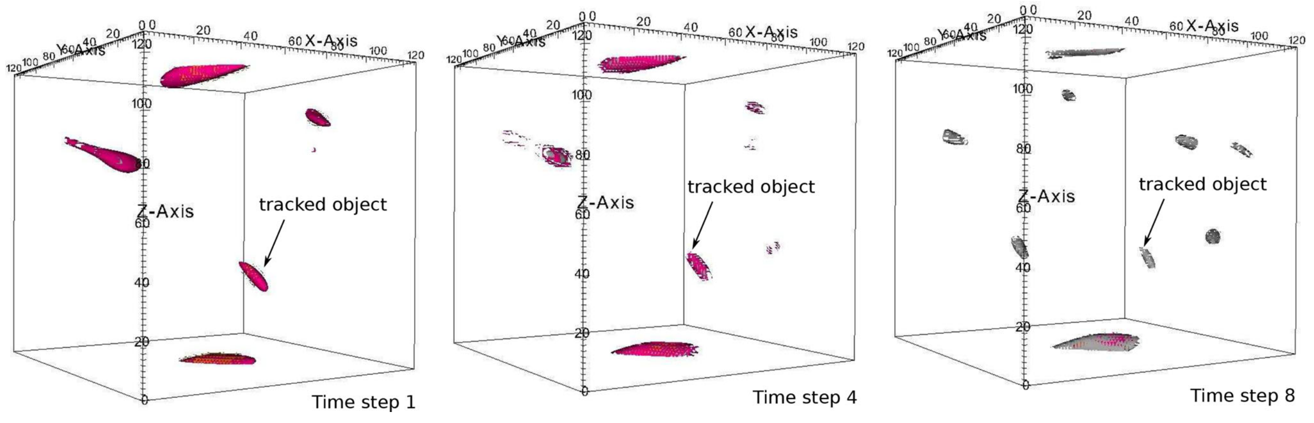


Fig. 11 A visualized view of the evolving volume regions (objects) tracked by our feature-based tracking algorithm.

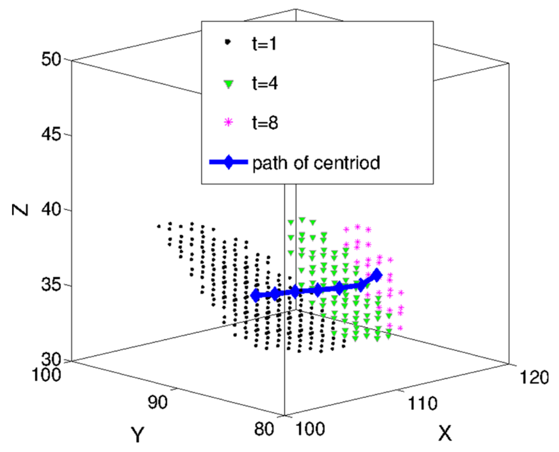


Fig. 12 Illustration of tracked path for object evolves over multiple time steps.

object in our experiments. In Fig. 12 we present the tracking of a object (the same one as in visualized Fig. 11), as seen by DOC, at three different time steps. As shown in the figure, this object moves from left to right and shrinks in size.

6 Related work

6.1 In-situ scientific data processing

The increasing performance gap between computing and I/O, and the cost of moving large volume of data to/from disks, motivates computation scientists to employ the in-situ data processing approach to perform analysis, visualization [15], indexing building [16], compression etc. The key idea is to move operations to data where the simulation is running. However, the existing inline approach [17], [18] tightly integrate analysis or visualization libraries into simulation code. Our techniques provide a more generic framework to compose and execute in-situ data operations in a flexible and customizable way.

6.2 Staging area based in-transit data analysis and I/O

The data staging area is a set of additional compute nodes allocated by users when launching the parallel simulations. The application of staging area has been investigated to add values to simulation's I/O pipeline in projects such as DataStager [19], PreData [20], JITStaging [21], ActiveSpace [22] and Glean [23]. Our techniques focus on scheduling and running analysis code in-situ to exploit increasing hardware parallelism and intra-node locality, and can be integrated with these in-transit approaches to perform hybrid data staging and analysis. Recent research [4] has also explored the benefits of combining both in-situ and in-transit approaches on high-end systems, and demonstrated the benefits of such hybrid approaches.

6.3 Tuple space model for coordination and communication

Our system framework provides the abstraction of a virtual shared data space in which application components of the in-situ data analysis workflow can put and get data. This concept of abstraction derives from the Tuple space model. Several academic and industrial projects are based on this model, like Linda [24], IBM Tspace , JavaSpace. Seine [25,26] builds a virtual shared space similar to our work, which is exclusively used for code coupling. DataSpaces [27] builds on Tuple space model a more general programming framework for data exchange and supports flexible asynchronous interaction patterns (e.g., publisher/subscriber/notification). In addition to support of shared data space abstraction, our techniques add the programming interface to compose in-situ data analysis workflow, and implement the runtime system with data-centric task mapping and more efficient shared memory based intra-node communication.

6.4 Clustering techniques

Guha et al. [28] provides a study on K-median clustering in order to model data streams. Providing a through explanation on the upper and lower bound of the algorithm is provided when a single pass is used on the data. The study thereby determines the total runtime and memory usage required for this algorithm to process a given data stream. This work serves as a base to understand the inherent problems faced by data analytics algorithms in order to process data in a continuous manner. Charikar et al. [29] enables k-means to cluster data streams efficiently by employing a divide and conquer method, which has a portion of the information processed offline and another as the data is created. Aggarwal et al. [30] address the problem of the poor quality of clusters when data evolves considerably over time. The approach performs both an online and offline analysis of the information. The online component periodically stores detailed summary statistics of the data being processed and the offline component is utilized to provide an understanding of the overall clusters in the data stream. Using these two methods together provides a framework where clusters are of a better quality than those taken at any one point in time. O'Callaghan et al. [31] provide a novel clustering algorithm that splits into two different algorithms STREAM and LOCALSEARCH, where the STREAM algorithm determines the amount of data to be processed and LOCALSEARCH is applied. Csernel et al. [32] use a sampling method as a means to reduce the total information needed to analyze data streams over time. They keep a portion of the cluster information so that they can view the data across time. They show that their sampling method does not affect accuracy and in most cases provide speedup.

6.5 Cluster tracking

Abrantes and Marques [33] studied the stochastic information within a data cluster over time in order to track the movement of cluster in the context of computer vision. Their approach proposes a noise model that can increase the robustness of the motion model with respect to outliers affecting the quality of motion detection. They show its effectiveness by evaluating car traffic sequences and ultrasounds. Abrantes and Marques also explored the idea of tracking clusters based on specific internal features of a cluster and demonstrated how cluster features can be used to identify distinct clusters at different time frames. We use this concept outside the area from which this method was created, computer vision.

6.6 Features tracking of time-varying simulation datasets

Many techniques have been developed by computer vision community to extract, identify and track features. Silver et al. [34] presented a semi-automatic volume tracking algorithm to improve visualization of 3D time-varying CFD datasets. Chen et al. [35] developed a parallel algorithm to analyze and visualize the evolving features extracted from time-varying simulation datasets in realtime. Our technique represents features as clustered data points in a multi-dimensional information space, and identifies and tracks the data clusters of interest in a distributed and timely manner using DOC.

7 Conclusion and future work

This paper explores the in-situ execution of feature-based objects tracking of time-varying scientific simulation data. Specifically, we present a feature-based cluster tracking algorithm which is built on DOC to process data in a distributed and timely manner. We also present Co-located DataSpaces, a framework and its programming interface, to compose and run tightly coupled workflow applications in-situ. We evaluated the proposed approach on the Lonestar cluster at TACC through different experiments. The experiments measured the end-to-end data transfer performance as well as the effectiveness and accuracy of our cluster tracking algorithm. The results show our in-situ memory-to-memory approach performs 10 times faster compare to traditional disk I/O. Moreover, our tracking algorithm achieves the tracking accuracy of 92.28 % on average.

Our direction for future work includes extending the decentralized in-situ cluster tracking system for other application areas such as online monitoring of resource utilization and anomaly detection in large scale data centers. We will also explore the use of on-node NVRAM/SSD storage to support energy-efficient in-situ staging of large data sets which could not be stored in current on-node memory.

Acknowledgments The research presented in this work is supported in part by US National Science Foundation (NSF) via Grants numbers OCI 1310283, DMS 1228203, IIP 0758566, OCI 1339036 and CNS 1305375, by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy through the Scientific Discovery through Advanced Computing (SciDAC) Institute of Scalable Data Management, Analysis and Visualization (SDAV) under ward number DE-SC0007455, the Advanced Scientific Computing Research and Fusion Energy Sciences Partnership for Edge Physics Simulations (EPSI) under award number DE-FG02-06ER54857, the ExaCT Combustion Co-Design Center via subcontract number 4000110839 from UT Battelle, and by an IBM Faculty Award. This work used the Extreme Science and Engineering Discovery Environment (XSEDE) under project number TG-CCR110035, which is supported by NSF grant number OCI 1053575. The research was conducted as part of the NSF Cloud and Autonomic Computing (CAC) Center at Rutgers University and the Rutgers Discovery Informatics Institute (RDI2). We thank Dr. Deborah Silver and Sedat Ozer for useful discussions on data visualization and providing the scientific dataset for our experimental evaluation.

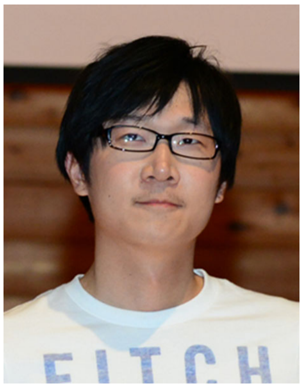
References

1. Childs, H.: Architectural challenges and solutions for petascale postprocessing. J. Phys. **78**(1), 12 (2007)
2. Gamell, M., Rodero, I., Parashar, M., Poole, S.: "Exploring energy and performance behaviors of data-intensive scientific workflows on systems with deep memory hierarchies". In: Proceedings of the 20th International Conference on High Performance Computing (HiPC), pp. 1–10. (2013)
3. Zhang, F., Docan, C., Parashar, M., Klasky, S.: "Dads: a dynamic and adaptive data space for interacting parallel applications". In: Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2010), Marina Del Rey (2010)
4. Bennett, J.C., Abbasi, H., Bremer, P.-T., Grout, R., Gyulassy, A., Jin, T., Klasky, S., Kolla, H., Parashar, M., Pascucci, V., Pebay, P., Thompson, D., Yu, H., Zhang, F., Chen, J.: "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis". In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ser. SC '12, 2012, pp. 49:1–49:9
5. Gamell, M., Rodero, I., Parashar, M., Bennett, J., et al.: "Exploring power behaviors and tradeoffs of in-situ data analytics". In: International Conferencce on High Performance Computing Networking, Storage and Analysis (SC), pp. 1–12. Denver, Nov 2013
6. Quiroz, A., Parashar, M., Gnanasambandam, N., Sharma, N.: "Design and evaluation of decentralized online clustering". ACM Trans. Auton. Adapt. Syst. **7**(3), 34:1–34:31 (2012). doi:10.1145/2348832.2348837
7. Quiroz, A., Gnanasambandam, N., Parashar, M., Sharma, N.: Robust clustering analysis for the management of self-monitoring distributed systems. Clust. Comput. **12**(1), 73–85 (Mar. 2009)
8. Chen, J.H., Choudhary, A., de Supinski, B., DeVries, M., Hawkes, E.R., Klasky, S., Liao, W.K., Ma, K.L., Mellor-Crummey, J., Podhorski, N., Sankaran, R., Shende, S., Yoo, C.S.: Terascale direct numerical simulations of turbulent combustion using s3d. Comput. Sci. Discov. **2**, 1–31 (2009)
9. Docan, C., Parashar, M., Klasky, S.: "Dataspaces: an interaction and coordination framework for-coupled simulation workflows". Clust. Comput. **15**(2), 163–181 (2012). doi:10.1007/s10586-011-0162-y
10. Podhorszki, N., Klasky, S., Liu, Q., Docan, C., Parashar, M., Abbasi, H., Lofstead, J., Schwan, K., Wolf, M., Zheng, F., Cummings, J.: "Plasma fusion code coupling using scalable i/o services and scientific workflows". In: Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science, ser. WORKS '09, pp. 8:1–8:9. ACM, New York, (2009) doi:10.1145/1645164.1645172
11. Pak, A., Paroubek, P.: "Twitter as a corpus for sentiment analysis and opinion mining". In: LREC, Baton Rouge (2010)
12. Zhang, F., Docan, C., Parashar, M., Klasky, S., Podhorszki, N., Abbasi, H.: "Enabling in-situ execution of coupled scientific workflow on multi-core platform". In: Proceedings of 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS'12), (2012)
13. Quiroz, A.: Decentralized online clustering for supporting autonomic management of distributed systems. Ph.D in Electrical and Computer Engineering, Rutgers University, (2010)
14. Schmidt, C., Parashar, M.: "Flexible information discovery in decentralized distributed systems". In: Proceedings of the 12th High Performance Distributed Computing (HPDC), pp. 226–235. (2003)
15. Yu, H., Wang, C., Grout, R., Chen, J., Ma, K.-L.: In situ visualization for large-scale combustion simulations. IEEE Comput. Graph. Appl. **30**(3), 45–57 (2010)
16. Kim, J., Abbasi, H., Chacon, L., Docan, C., Klasky, S., Liu, Q., Podhorszki, N., Shoshani, A., Wu, K.: "Parallel in situ indexing for data-intensive computing". In: Proceedings of IEEE Symposium on Large Data Analysis and Visualization (LDAV'11), Oct (2011)
17. Whitlock, B., Favre, J.M., Meredith, J.S.: "Parallel in situ coupling of simulation with a fully featured visualization system". In: Proceedings of 11th Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'11), Apr (2011)
18. Fabian, N., Moreland, K., Thompson, D., Bauer, A., Marion, P., Gevecik, B., Rasquin, M., Jansen, K.: "The paraview coprocessing library: a scalable, general purpose in situ visualization library". In Proceedings of IEEE Symposium on Large Data Analysis and Visualization (LDAV'11), Oct (2011)
19. Abbasi, H., Wolf, M., Eisenhauer, G., Klasky, S., Schwan, K., Zheng, F.: "Datastager: scalable data staging services for petascale applications". In: Proceedings of 18th International Symposium on High Performance Distributed Computing (HPDC'09), (2009)
20. Zheng, F., Abbasi, H., Docan, C., Lofstead, J., Klasky, S., Liu, Q., Parashar, M., Podhorszki, N., Schwan, K., Wolf, M.: "PreDatA - preparatory data analytics on peta-scale machines". In: Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS'10), Apr (2010)
21. Abbasi, H., Eisenhauer, G., Wolf, M., Schwan, K., Klasky, S.: "Just in time: adding value to the IO pipelines of high performance applications with JIT staging". In: Proceedings 20th International Symposium on High Performance Distributed Computing (HPDC'11), June (2011)
22. Docan, C., Parashar, M., Cummings, J., Klasky, S.: "Moving the code to the data - dynamic code deployment using active spaces". In: Proceedings of 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS'11), May (2011)
23. Vishwanath, V., Hereld, M., Papka, M.: "Toward simulation-time data analysis and i/o acceleration on leadership-class systems". In: Proceedings of IEEE Symposium on Large Data Analysis and Visualization (LDAV'11), Oct 2011
24. Gelernter, D.: Generative communication in Linda. ACM Trans. Programm. Lang. Syst. **7**(1), 80–112 (1985)
25. Zhang, L., Parashar, M.: "A dynamic geometry-based shared space interaction framework for parallel scientific applications". In: Proceedings of the 11th International Conference on High Performance Computing (HiPC'04), 2004
26. "Enabling efficient and flexible coupling of parallel scientific applications". In: Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS'06), 2006
27. Docan, C., Parashar, M., Klasky, S.: "DataSpaces: an interaction and coordination framework for coupled simulation workflows". In: Proceedings of 19th International Symposium on High Performance and Distributed Computing (HPDC'10), June 2010
28. Guha, S., Meyerson, A., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams: theory and practice. IEEE Trans. Knowl. Data Eng. **15**(3), 515–528 (2003)
29. Charikar, M., O'Callaghan, L., Panigrahy, R.: "Better streaming algorithms for clustering problems". In: Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing, pp. 30–39. (2003)
30. Aggarwal, C.C., Watson, T.J., Ctr, R., Han, J., Wang, J., Yu, P.S.: "A framework for clustering evolving data streams". In: VLDB, pp. 81–92. (2003)
31. O'Callaghan, L., Mishra, N., Meyerson, A., Guha, S., Motwani, R.: "Streaming-data algorithms for high-quality clustering". In: 2013 IEEE 29th International Conference on Data Engineering (ICDE) pp. 0685-0685. IEEE Computer Society (2013)
32. Csernel, B., Clerot, F., Hbrail, G.: "Streamsamp: datastream clustering over tilted windows through sampling". In: ECML PKDD 2006 Workshop on Knowledge Discovery from Data Streams, (2006)

33. Abrantes, A.J.,Marques, J.S.: “A method for dynamic clustering of data”. In: British Machine Vision Conference, (1998)
34. Silver, D., Wang, X.: Tracking and visualizing turbulent 3d features. IEEE Trans. Visual. Comput. Graph. **3**(2), 129–141 (1997)
35. Chen, J., Silver, D., Parashar, M.: “Real-time feature extraction and tracking in a computational steering environment”. In: Proceedings of Advanced Simulations Technologies Conference (ASTC’03), (2003)

Solomon Lasluisa is a PhD student at Rutgers University. He received his BS and MS degrees in Electrical and Computer Engineering at Rutgers University. His research interests include data analytics, cloud computing, and wireless networks. His research activities include autonomic error detection for data centers to prevent instability due to hardware errors in scientific simulations.

Fan Zhang received the BS and MS degrees in Computer Science from Huazhong University of Science and Technology, China. He is currently working towards the PhD degree at the Department of Electrical and Computer Engineering at Rutgers University. His research interests include parallel and distributed computing, high-performance computing and scientific data management. Contact him at zhangfan@cac.rutgers.edu.



Tong Jin is a Ph.D candidate and research assistant in the Department of Electrical and Computer Engineering and the US National Science Foundation (NSF) Cloud and Autonomic Computing Center (CAC) at Rutgers University. His research interests are broadly in big data management, distributed and parallel computing, high performance computing, cloud computing, and distributed wireless networked systems. He got his MS degree majoring in Computer Engineering at Rutgers University. He is a student member of IEEE and ACM. Contact him at tjin@rutgers.edu.

Ivan Rodero is an Assistant Research Professor at the Rutgers Discovery Informatics Institute (RDI2) and the US National Science Foundation (NSF) Cloud and Autonomic Computing Center (CAC) at Rutgers. His research interests include parallel and distributed computing, high-performance computing, energy efficiency, autonomic computing, grid computing, cloud computing, and big data. He has a PhD in computer science and engineering from the Technical University of Catalonia, Spain. Ivan Rodero is recipient of the 2014 IEEE TCSC Award for Excellence for Early Career Researchers. He is a senior member of IEEE, and a member of ACM, and the American Association for the Advancement of Science (AAAS). Contact him at irodero@rutgers.edu.

Hoang Bui received the B.S. and M.S. in Computer Science in 2004 and 2007 from Midwestern State University and PhD in Computer Science and Engineering in 2012 from the University of Notre Dame. He is currently a Postdoctoral Associate of RDI2 at Rutgers University, where his research focuses on optimizing data management for scientific applications running on high-end computing platforms.

Manish Parashar is Professor of Computer Science at Rutgers University. He is also the founding Director of the Rutgers Discovery Informatics Institute (RDI2) and site Co-Director of the NSF Cloud and Autonomic Computing Center (CAC). His research interests are in the broad areas of Parallel and Distributed Computing and Computational and Data-Enabled Science and Engineering. Manish serves on the editorial boards and organizing committees of a large number of journals and international conferences and workshops, and has deployed several software systems that are widely used. He has also received a number of awards and is Fellow of AAAS, Fellow of IEEE/IEEE Computer Society and Senior Member of ACM. For more information please visit http://parashar.rutgers.edu/.