A real-time machine learning and visualization framework for scientific workflows

Feng Li*
Indiana University-Purdue University
Indianapolis, Indiana
lifen@iupui.edu

Fengguang Song†
Indiana University-Purdue University
Indianapolis, Indiana
fgsong@iupui.edu

ABSTRACT

High-performance computing resources are currently widely used in science and engineering areas. Typical post-hoc approaches use persistent storage to save produced data from simulation, thus reading from storage to memory is required for data analysis tasks. For large-scale scientific simulations, such I/O operation will produce significant overhead. In-situ/in-transit approaches bypass I/O by accessing and processing in-memory simulation results directly, which suggests simulations and analysis applications should be more closely coupled. This paper constructs a flexible and extensible framework to connect scientific simulations with multi-steps machine learning processes and in-situ visualization tools, thus providing plugged-in analysis and visualization functionality over complex workflows at real time. A distributed simulation-time clustering method is proposed to detect anomalies from real turbulence flows.

CCS CONCEPTS

• **Computing methodologies** → **Parallel computing methodologies**; *Modeling and simulation*; Unsupervised learning; • **Software and its engineering** → *Software organization and properties*;

KEYWORDS

High performance computing, DataSpaces, Computational fluid dynamics, Real-time data analysis, Scientific workflow

ACM Reference Format:
Feng Li and Fengguang Song. 2017. A real-time machine learning and visualization framework for scientific workflows. In *Proceedings of PEARC17*. ACM, New York, NY, USA, 8 pages. https://doi.org/http://dx.doi.org/10.1145/3093338.3093380

1 INTRODUCTION

Recent advancements in parallel computing and high performance computing (HPC) have given scientists the power to solve extremely large problems that were impossible to tackle before. Reports from Boeing [5, 13] depict important user cases that with more powerful supercomputers, numerical simulation of various approximations to the Navier-Stokes equation can significantly reduce the testing cost by increasing the scale and computation accuracy during the development of new Boeing aircrafts.

Typical real world numerical simulations not only involve intensive computations but also generate huge amounts of data. For instance, S3D, a massively parallel DNS (direct numerical simulation) solver developed at Sandia National Laboratories, requires extreme-scale computing power and outputs tens of terabytes of data [11]. In an S3D workflow, 30-130TB of data is generated per simulation and the data must be first archived to local disks, and then moved to distributed storage systems to make enough space for the subsequent time steps [6]. Thus, further data analysis of the simulation results includes: 1) loading massive amounts of data from distributed storage systems; 2) reading data to the active disks; and 3) loading data into main memory for analysis. However, the expensive I/O overhead involved in the above procedure can be alleviated by using an in-situ (or in-memory) processing approach.

In an in-situ approach, analysis applications are directly interacting with simulation applications (it is common that different applications are recompiled, linked together, and run in the same process). This approach is able to totally avoid the I/O overhead mentioned above since the analysis applications can directly read data from the same main memory shared with the simulation applications. There are several limitations with this approach:

(1) Simulation and analysis applications are in the same address space, which means simulation developers and analysis/visualization developers have to closely work together to carefully define how to integrate two open source codes.
(2) Whenever a user wants to modify analysis applications or switch to different analysis methods, the whole application must be stopped, rebuilt, and restarted.
(3) Analysis and simulation applications may have different scaling behaviors, and binding them together will bring issues of load balancing to both applications.
(4) Most often data-intensive analysis applications are memory-demanding, and thus cannot fit into the same compute node as the simulation application.

To combat these issues, we use a *tuple space* to connect different applications so that analysis or visualization specialists no longer need to know the details of simulation code and vice versa. This way, simulation and analysis applications are decoupled from each other such that different analysis tasks can be used and substituted at runtime. In the tuple-space based approach, data communication

*PhD student at IUPUI.
†Assistant professor in Department of Computer Science at IUPUI.

PEARC17, July 09-13, 2017, New Orleans, LA , USA

ACM ISBN 978-1-4503-5272-7/17/07…$15.00
https://doi.org/http://dx.doi.org/10.1145/3093338.3093380

is realized through Remote Direct Memory Access (RDMA). In this paper, we will describe the following contributions in details:

(1) Demonstration of a real-time workflow system that includes simulation, real-time data analysis and visualization components targeting the domain of computational fluid dynamics (CFD) and turbulence analysis.
(2) Design and implementation of an interactive software infrastructure where different types of applications can efficiently cooperate using a tuple space.
(3) Design of a new scalable *parallel non-parametric anomaly detection algorithm* that serves as the real-time machine learning analysis method for CFD turbulence analysis.

The rest of the paper is organized as follows. Section 2 introduces the background of DataSpaces, in-situ visualization techniques, and vortex detections in turbulence flows. Section 3 presents the related work to couple applications in scientific workflows. Section 4 describes our new parallel machine learning algorithm for anomaly detections. Section 5 presents the design and implementation of our software framework. Section 6 shows the experimental results, and Section 7 summarizes the work.

2 BACKGROUND

2.1 Tuple Space and DataSpaces

Tuple space is an associative memory model that is intended for high-productivity and distributed/parallel computing. Within the model, *tuples* are accessed by content and type, rather than by their raw memory addresses. The strength of this model is its ability to describe data without referencing to any specific computer architecture. By using the tuple space model, different simulation and analysis applications can be flexibly combined to reveal insights to users dynamically.

DataSpaces [8] follows the tuple space model, and builds a flexible interaction and coordination substrate for various applications so that they can interact at runtime. The live data from simulation is extracted from a simulation application, and can be indexed and accessed by other applications using semantically meaningful operators. In a typical simulation workflow, different types of coupled jobs are launched at large scales, and thousands of compute nodes are run to generate significantly large amounts of data. Also, different applications will exchange data frequently. These interactions would become overly complicated and varied provided using a low-level message passing programming model. DataSpaces solves this problem by providing a flexible, simple, high-level abstraction of a semantically specialized, distributed and virtual shared space that allows concurrent accesses by various applications.

In this work, we use DataSpaces to combine computational fluid dynamics (CFD) simulations with machine learning analysis and visualization automatically.

2.2 Paraview Catalyst and In-Situ Visualization

Large scale simulations may take weeks or even months to compute a great amount of output data. On the other hand, scientists often want to detect or look into specific phenomena while the simulations are running so that the simulation can be paused, or terminated in advance if unusual patterns are found during analysis.

In situ (or in-memory) data analysis is a widely used approach to analyzing data while the data still resides in memory. Instead of outputting simulation data to secondary storage, analysis is performed in memory while data is being produced.

Paraview Catalyst (also known as Paraview Coprocessing Library) [10] is one of the early attempts to visualize large scale data sets. It provides an adaptable API (application programming interface) between simulation and visualization tasks. Unlike specialized systems such as the hurricane prediction visualization [9], Paraview Catalyst provides a generic in-situ visualization framework.

In order to instruct a simulator at runtime, Paraview Catalyst requires developers to implement only three routines: *Initialize*, *Coprocess*, and *Finalize*. The Coprocess routine will be invoked to convert simulation raw data and perform different types of analysis and visualization in each time step.

In this work, we use the Paraview Catalyst library to realize online CFD visualizations at real time.

2.3 Application of Vortex Detections in Turbulence Flow

Besides computation-intensive simulations, this paper also targets big data analysis to look for interesting features that can be regarded as patterns occurring in the dataset. Unlike a lot of features that can be defined precisely, some features and patterns are quite common but don't have precise definitions. Vortex is one of such features in flow fields. Hence, our work focuses on the important CFD vortex detection problem at real time.

In general, a vortex is characterized by the swirling motion of fluid around a central region. A few taxonomies have been proposed to classify vortex detection methods [12]. How to define a vortex is one of those taxonomies. The vortex-finding methods can be classified into region-based or line-based. Region-based vortex detection is to identify whether continuous cells belong to a vortex; while line-based vortex detection is to identify vortices by locating vortex core lines. In practice, region-based methods are easier to implement and are less computationally expensive.

In this paper, we design and develop a region-based vortex detection algorithm using a non-parametric divergence estimation method (more details are provided in Section 4).

3 RELATED WORK

There are many data and computational services, which are largely shared and distributed [21]. However, the actual access and deployment on different systems often reduce the productivity. Workflow is then used to to simplify coupling different applications.

For instance, Kepler is a widely used workflow framework where scientists, analysts, and software developers can share data and models over Internet (via Web Services) [17]. Kepler provides a graphical user interface (GUI) where users can simply select and connect different data sources and analytical components to create a scientific workflow. Typical scientific workflows involve jobs that are data-intensive, computation-intensive, and visualization intensive. Web service extensions are needed so that scientific workflows can access remote resources and services seamlessly.

Pegasus [7] is another widely used workflow framework that can map complex scientific workflows to distributed resources. But

in workflow frameworks such as Kepler and Pegasus, they typically use files and slow disk I/Os to exchange data between applications. They also require querying file existence to detect if new data is available or not.

ADIOS (Adaptive I/O System) [15] is designed to support an arrange of data transfer methods for integrating workflow components. Designed by ORNL and Georgia Institute of Technology, ADIOS enables scalable, portable and efficient componentization of the I/O layer on both Linux clusters and supercomputers. It also provides I/O componentization for different data transport methods, which makes switching I/O methods in different infrastructures simpler. Instead of parsing user-input arguments, ADIOS takes an external XML file as configuration, which is portable in different environments. With this design, changing I/O routines in application code is as simple as editing a single entry in the XML file.

ADIOS provides DataSpaces as a data transport method (implemented with a wrapper). DataSpaces can provide low-overhead, high-throughput data extraction from running simulations. Our framework uses DataSpaces to integrate CFD simulations with machine learning applications. Unlike ADIOS, our system is a high-level application-specific framework, which is focused on simulation-time CFD anomaly detections.

4 AN UNSUPERVISED MACHINE LEARNING ALGORITHM TO ANALYZE CFD FLOWS

In certain data science domains, it is relatively simple to view data points as fixed, finite-dimensional features. There is another class of domains, which assumes that data corresponds to a continuous probability distribution, as mentioned in [19]. In the second domain, each group of data can be regarded as independent and identically distributed (i.i.d.) samples, and k-nearest neighbors (kNN) statistics are required to obtain divergence between data groups. Once we know the divergence (or difference) of all region pairs, regions can be clustered into different categories.

To compute a divergence, we employ the non-parametric divergence estimation from [19]. The basic idea is as follows: Each group of data $X_{1:n} = (X_1, ..., X_n)$ is regarded as n i.i.d samples from a distribution with density p. If there is another group of data $Y_{1:m} = Y_1, ..., Y_m$ from another distribution with density q, The difference between X and Y can be calculated by using a divergence estimator. A common divergence metric is the L_2 divergence, which is described in the following definition:

Definition 4.1. Let p and q be densities over R^d, then the L_2 divergence is:

$$L(p||q) = \int (p(x) - q(x))^2 dx)^{1/2}.$$

Yet we still need to know the densities of distributions such as $p(x)$ and $q(x)$.

The densities of $p(x)$ and $q(x)$ can be estimated with the kNN methods [16] . Given two data groups $X_{1:n}$ and $Y_{1:m}$. Let $v_k(i)$ be the Euclidean distance from X_i to its k-th nearest neighbor in $X_{i:n}$. Similarly, let $r_k(i)$ be the distance from Y_i to its k-th nearest neighbor in $Y_{1:n}$. We use c to represent the volume of a d-dimensional unit ball. The density of $p(x)$ at X_i given a parameter of k can be expressed as follows:

$$p_k(X_i) = k/((n-1) \times c \times v_k(i))$$

Similarly, the density of $q(x)$ at Y_i given a parameter of k is:

$$q_k(Y_i) = k/(m \times c \times r_k(i)).$$

To apply the non-parametric divergence estimation to our problem, we describe each point in a fluid region using three dimensions: v_x and v_y for the point's velocity in x and y directions, and dc for the point's distance to the fluid region's center. The dc dimension is chosen since the velocity in a vortex most often changes with the distance from the center, as described in [20].

Next, we use the k-medoids clustering method to cluster all regions using the above divergence estimation. As a classical partitioning technique, k-medoids is robust to noise and outliers. It takes dissimilarities between all elements as input and minimizes the sum of in-cluster dissimilarities. Since divergence is also a metric of difference between regions, we use it to construct the dissimilarities matrix and feed it to the k-medoids method. We run k-medoids for multiple times, each time with random initial medoids. The clustering result that has the best in-cluster dissimilarities is the output. The algorithm is described in Algorithm 1.

Algorithm 1: Clustering based on Non-parametric Divergence Estimation (*npdivs*)

Input:
Regions of data points $R_{1:n}$, each point has (v_x, v_y, dc)
k, number of clusters
num_runs, number of runs of k-medoids method
Result:
cluster_ids, cluster id of each region
Begin
Estimate all pairwise divergences:
$divs(R_1, R_2), divs(R_1, R_3)...divs(R_{n-1}, R_n)$ among all regions from $R_{1:n}$ using *npdivs* method ;
Construct a distance matrix D from
$divs(R_1, R_2), divs(R_1, R_3)...divs(R_{n-1}, R_n)$;
// run k-medoids for multiple times
for $i = 0; i < num\_runs; i++$ **do**
  randomly choose k initial medoids ;
  run k-medoids until sum of in-cluster dissimilarity doesn't change ;
  update *cluster_ids* if smaller sum of in-cluster dissimilarity is achieved in this run ;
end
// return best clustering results
return *cluster_ids* ;
End

Note that a direct implementation of the algorithm cannot scale well with large datasets since computing the pairwise divergences among all regions can be extremely expensive. Hence, we present an improved approach to solving this issue (we present details in Section 5.3).

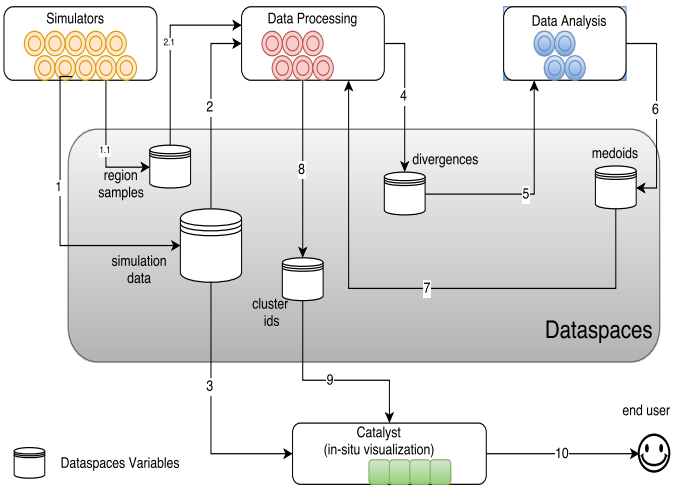

Figure 1: Architecture of the software framework.

5 SOFTWARE FRAMEWORK

Our integrated simulation and machine learning framework consists of four major components, which are the following independent applications: *Simulators*, *Data Processing*, *Data Analysis*, and *Catalyst visualization*. All the communication among applications is fulfilled through *DataSpaces*, and no file I/O is involved. Figure 1 shows the software system's architecture.

In the framework, *Simulator* flushes computed results into DataSpaces in each time step. The simulation result is split into strips and fetched by both *Data Processing* and *Catalyst* applications. Next, the *Data Processing* application works with the *Data Analysis* application to identify special patterns from stripped data; At the same time, *Catalyst* generates all required visualization data structures to provide users with visualization for both turbulence flow data (i.e., path 3) and data analysis results (i.e., path 9).

5.1 Simulators

In our work, the simulation results are generated in two ways: 1) The archived simulation data from a turbulence database [2]; and 2) the real-time simulations using CFD software (e.g., OpenFoam [4]).

Turbulence database: The Johns Hopkins Turbulence Database (JHTDB) provides web services with C/Matlab/python interfaces, and data cutout downloads for various types of turbulence datasets. We use a 2D cutout from the isotropic-coarse dataset as our first simulation data source. More details about the isotropic-coarse dataset is provided in Section 6.

Real-time numerical simulations: OpenFoam is an open source CFD software widely used in both engineering and science domains. It provides a large range of solvers for different problems. To run a simulation, a *solver* and a *case* need to be specified.

Different solvers reflect different algorithms or physical models. On the other hand, a *case* is a collection of files to describe the desired running conditions (such as mesh shape, start/end time, parameters for solvers) of the application. We use *icoFoam* as an example solver. IcoFoam solves incompressible laminar Navier-Stokes equations using the PISO algorithm.

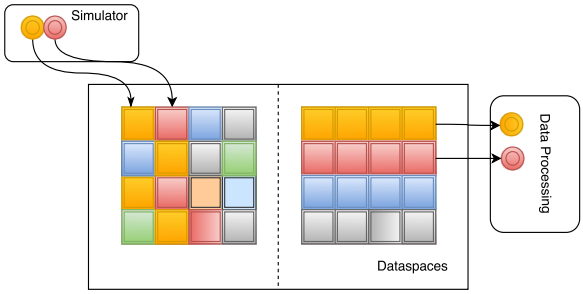

Figure 2: Data layout of simulation data in DataSpaces

To utilize DataSpaces, we write a new customized solver based on the original icoFoam, and overwrite its default I/O routine. The *2-D Lid-driven cavity flow* problem is used in our experiments. The problem is a well-known benchmark for incompressible fluid flows. Our simulation includes a fluid contained in a square container, which has three stationary sides, and one moving side with a constant velocity.

5.2 Data Processing and Data Analysis

As shown in Figure 1, our data analytics workflow involves the *Data Processing* application and the *Data Analysis* application. The two applications work as follows: 1) Simulation data is first divided into regions and distributed to different *data processing* processes; 2) *data processing* processes then read the assigned regions and compute the divergences between all pairs of regions; 3) After step 2 is completed, *data analysis* processes will read the computed divergences and perform k-medoids clustering to search for the k global medoids, where k is the number of clusters specified by users; and 4) the *data processing* processes assign a cluster ID to each of its allocated regions based on the region's distance (i.e., divergence) to the k medoids.

The rest of the subsection provides more details about the data processing and data analysis applications.

5.2.1 Data Processing. Since the produced simulation data may be in a large scale, we strip data into a number of *data processing* processes as shown in Figure 2. We partition data horizontally instead of blocks because it can achieve better spacial locality during DataSpaces operations. In our design, each *data processing* process can run without data dependency on each other. Synchronization between the data processing application and the next data analysis application is supported by customized DataSpaces locks (details will be discussed in Section 5.6).

5.2.2 Data Analysis. Once the *data analysis* application reads the divergences computed from the *data processing* processes, it will construct the distance matrix in local memory and perform k-medoids clustering for a number of times and return the medoids information when the smallest cost is achieved. The medoids information will later be sent back to the *data processing* processes so that they can assign a cluster ID to every region.

5.3 Distributed Sampling

However, for large scale problems, the pair-wise divergence calculation in the Data Processing application can be time consuming. We design a distributed sampling method to provide an approximation of the large simulation data. This method greatly reduces the CPU time of those *data processing* processes.

The original k-medoids algorithm works well for a small dataset, but does not scale when taking large datasets [18]. In our specific problem, suppose there are n fluid points (each point contains physical properties such as velocity, pressure, etc.), and those points are evenly divided into geometrically continuous regions of size $size\_region$. This leads to a number of $\frac{(n/size\_region)(n/size\_region-1)}{2}$ region pairs. Hence, the time complexity to compute pairwise divergences for all regions will be $O(n^2)$, where $size\_region$ is eliminated since it is a constant for a given resolution. Consequently, this quadratic complexity significantly slows down the entire workflow for a large input size.

To reduce the execution time for the pairwise comparisons (divergence calculation, in our case) in the k-mediod method, one promising approach called CLARA (Clustering LARge Applications) [14] was used to draw samples from the original dataset, then find medoids of the samples, and assign a cluster ID to all points in the original data set based on these medoids. Since samples are randomly drawn, medoids of the sampling can be considered an approximation of the medoids of the whole data set. Experiments in [14] have also shown that CLARA can achieve higher efficiency than the original1 k-medoids algorithm.

Since the *data processing* application has access to all simulation results (through path 2 in figure 1), the simplest method is to let *data processing* processes get the information of each random region sample through one DataSpaces 'get' operation (in path 2 in figure 1). However, such small-granularity operations are expensive and not recommended by DataSpaces because every single operation will involve a nontrivial amount of transmission and synchronization overhead. Therefore, a better method is to reorganize the data such that each single DataSpaces operation will put/get a large chunk of data.

In this paper, we design and develop a novel approach called *Distributed Sampling* to let each *simulation process* decide its own local samples randomly, and let DataSpaces aggregate samples from all the *simulation* processes. Figure 3 shows how the distributed sampling approach works. In each time step, local samples of regions are drawn by each simulation process (aka "Simulator"), and are gathered in the central DataSpaces. After DataSpaces gets samples from all the simulation processes, each *data processing* process can easily retrieve a copy of the set of samples. By splitting global sampling into: 1) local sampling, 2) aggregation of samples in DataSpaces, and 3) each process reads a copy of all samples, the access time to DataSpaces is greatly reduced. In particular, with this approach, a single `put` operation can incorporate the samples from each *simulator* process into a global continuous DataSpaces area while a single `get` operation can read all the samples at once.

5.4 In-situ Visualizations with Catalyst

We employ the visualization software of Paraview Catalyst to enable real-time visualizations of both CFD simulations and clustering analysis results. Paraview Catalyst can convert the simulation output to Paraview-dependent data structures, which are then fetched and visualized by Paraview servers.

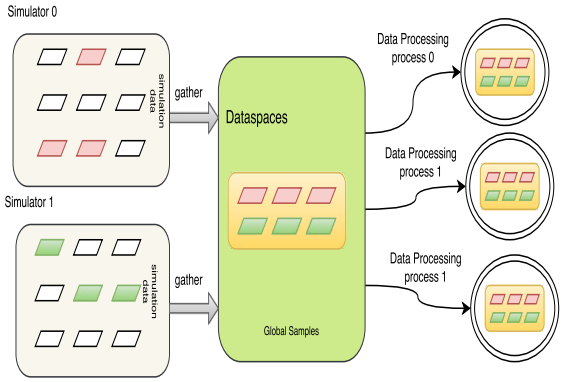

Figure 3: Distributed Sampling Using DataSpaces

As an end user, one can connect a laptop computer to Paraview servers directly by installing a Paraview GUI. Typically, using *Catalyst* includes the following two steps:

(1) *Pre-process* step. User needs to configure and specify visualization methods and parameters before simulation runs.
(2) *Execution* step. Simulation code is linked with Catalyst, and then generates visualization data structure at real time.

In the pre-process step, a python script is used to configure Catalyst on how visualization pipelines will be constructed and how data will be extracted (frequency and format) during the simulation. Instead of explicitly writing this script from scratch, the "Catalyst Script Generator plugin" allows users to generate scripts from templates and sample simulation data using the client-side Paraview GUI. To get the sample simulation data, the user can use the preconfigured scripts shipped with the Catalyst package, in which file writers are automatically generated for different visualization structures (VTK objects). A "live connection" option can also be enabled so that the visualization data are extracted to Paraview servers or directly-attached Paraview clients.

5.5 General DataSpaces Adapter

All applications interact with each other using the DataSpaces routines. For different types of simulation sources and analysis procedures, these routines can be reused easily to fit in various situations using a consistent interface. An example is shown in Listing 1. "put_common_buffer" is not shown here since it is nearly the same as "get_common_buffer".)

Listing 1: Interface of the Customized DataSpaces Adaptor

```
void get_common_buffer(
        int timestep,
        int bounds[6],
        int rank,
        MPI_Comm * p_gcomm,
        char * var_name,
        float **p_buffer,
        size_t elem_size,
```

```
double *p_time_used
);
```

To apply this interface to different variables used by a collection of coupled applications, users only need to specify the name for each variable (i.e., *var_name*), size of each element to be written (i.e., *elem_size*) and the pointer to the prepared receive buffer (i.e., *p_buffer*). The parameter of *bounds*[] corresponds to the geometric district to which the buffer will be written. In each time step, this interface is called by all processes of the same application.

5.6 Inter-Application Synchronizations using DataSpaces

It is desirable and helpful for users that both turbulence flows and data analysis results can be visualized side-by-side in real time. This requires each application coordinate with other applications in a pipplline way. Whenever a piece of simulation data is written to DataSpaces, it must be immediately consumed by both *Catalyst* and *Data Processing* applications. This gurantees the output shown to the users is not delayed. When there is an anomaly pattern appearing in the clustering analysis results, users can quickly compare it with the corresponding flow visualization.

We adopt the customized lock implemented in DataSpaces, which enforces writer/reader synchronizations. With this customized lock, writer applications can always acquire a write lock while calling *dspaces_lock_on_write* firstly. The subsequent calls to *dspaces_lock_on_write* require all read applications have fetched their data and released the corresponding read lock. However, using this collective lock directly will lead to poor performance. For instance, a simulation process may be blocked when it tries to acquire a write lock for which the corresponding consumer processes have not released all the read locks.

To solve the problem, we create a new lock for each time step's output (instead of using a single lock for one DataSpaces variable). This way we are able to make sure synchronizations among applications are still maintained in each time step. At the same time, the blocking between two consecutive times teps can be avoided. In fact, DataSpaces is able to store multiple versions (or time steps) of a variable, and the older versions are invalidated in the First In First Out (FIFO) manner.

6 EXPERIMENTAL RESULTS

We evaluate our computational framework using the Karst computer system located at the Indiana University ([3]). Karst is a high-throughput computer system, which has 228 general access compute nodes and 28 condo nodes, as well as 16 dedicated data nodes for data-intensive operations. Each compute node is an IBM NeXScale nx360 M4 server. Details of the compute node are presented in table 1. All the compute nodes have installed Red Hat Enterprise Linux 6 and are connected via 10-gigabyte Ethernet.

Our performance evaluation consists of three different experiments. The first experiment shows how our framework can support real-time flow visualization and real-time clustering-based anomaly detections. The second experiment analyzes the communication overhead and efficiency of our framework. The third experiment demonstrates the speedup and scalability of the framework.

Table 1: The Karst System.

Information	Per compute node
CPU	Two Intel Xeon E5-2650 v2 eight-core processors
#cores	16
Main memory	32GB
Secondary storage	250GB

6.1 Turbulence Analysis with JHTDB Dataset

Our first experiment was performed with the forced isotropic dataset [1] from Johns Hopkins Turbulence Databases (JHTDB). A pseudo-spectral method was used to solve a direct numerical simulation on a $1024 \times 1024 \times 1024$ grid. Original isotropic dataset contains 5028 frames of data, which includes velocity and pressure. This dataset was chosen because the data is well formatted in VTK/HDF5 so that we can quickly verify analysis results using visualization tools. Note that both "fine" and "coarse" isotropic data are provided by the original datasets from JHTDB. Our experiment uses the "coarse" dataset in which frames are stored in every 10 time steps.

We use a slice of size 200×200 from the isotropic dataset with a simulation duration of 100 frames. To simulate the real-time turbulence dataflow, we use the the HDF5 reader which can extract both velocity and pressure information from the dataset downloaded from JHTDB periodically.

Figure 4 shows a snapshot of our real-time data analysis results. The whole area is divided into 20×20 square regions. Divergences are then calculated for every pair of regions. By using the k-medoids clustering algorithm presented in Section 4, each region is assigned with a cluster ID. The right part of the demo shows flow properties in the current time step, with the colors and arrows representing pressure and velocity, respectively. The left part is the real-time output of our clustering analysis. The colors of different blocks show which category the current region belongs to. In this example, all regions are clustered into one of the three categories below: (1) steady flow, where fluid in this region has similar velocity and moves steadily in the same direction, (2) unsteady flow, where swirling motions of fluid can be found, and (3) random flow, where velocity of fluid change irregularly over time.

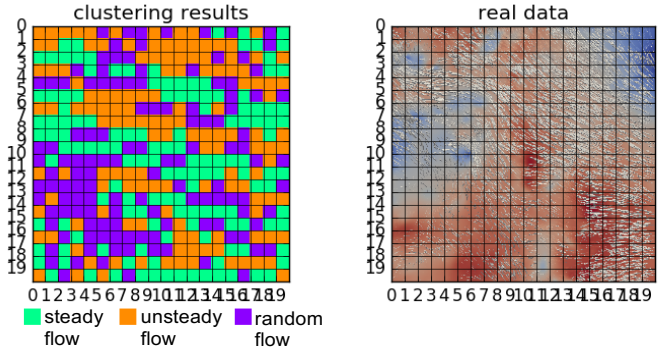

Figure 4: Demo on the JHTDB isotropic dataset

As presented in Section 5, a frame of simulation data is sent to DataSpaces in each time step, and then fetched by both *data processing* and *Catalyst* applications. Finally, both clustering results and turbulence flow properties (i.e., velocity and pressure) are processed by the *Catalyst* application. The Catalyst visualization results can be shown in a client-side Paraview GUI. In Paraview, two views that use the same grid and cameras are linked together so that users can easily find the mappings from clusters to turbulence flow.

6.2 Execution Time Breakdown

The second experiment is used to show the efficiency and communication overhead of our framework. Since there are multiple applications in our framework and all of them use DataSpaces to communicate with each other, it is important to examine how much time is spent on each interaction. We study the software efficiency by breaking down the total execution time.

The configuration of our workflow experiments is shown in Table 2. We use 16 simulation processes, each of which computes and generates a set of fluid velocity and pressure data for a $1024x1024$ grid. We also use 32 *data processing* processes, each of which reads a strip of simulation data from DataSpaces, then calculates the divergences, and finally gets the k-medoids information from the *analysis* processes.

Figure 5 shows the average data transfer and computation time of all the four applications for 30 time steps: *simulator*, *(data) processing*, *(data) analysis*, and *catalyst*. From the figure we can make the following observations:

Table 2: Configuration for the time-breakdown experiment.

simulation processes	16
fluid grid size/process	1024x1024
data processing processes	32
data analysis processes	1
Catalyst processes	1
region size	16x16
number of clusters (k)	3
max number of versions buffered	30

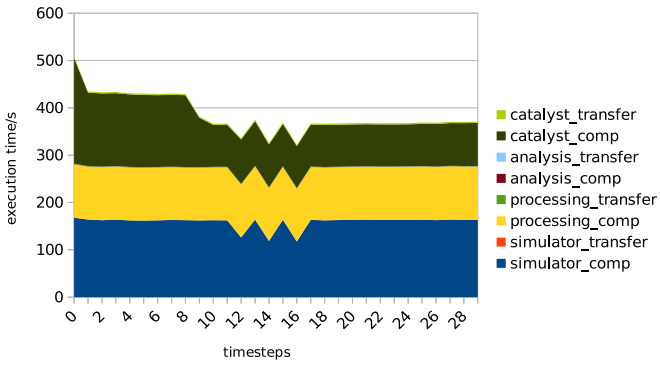

Figure 5: An execution time breakdown

(1) In all the four applications, data transfer time is much less than computation time.
(2) *Simulator* application has the highest computation time among all the four applications. It means under the current configuration, *simulator* won't be delayed by other applications, thus the entire workflow will have a small end-to-end latency.
(3) The computation time of different applications are in a similar magnitude (except for the analysis application, which is not as computation intensive as the other applications and uses only one process). This way all computing resources are kept busy during the entire simulation.

6.3 Scalability Evaluation

We also want to study how well the workflow system can scale with more computing resources. In this experiment, we use a fixed problem size and increase the number of processes for various applications.

The end-to-end execution time is measured from the beginning of the simulation step to the end of the catalyst co-processing step. We also measure the elapsed time of each application. Table 4 shows the time to compute an input of size 4096×4096, which contains 1GB of velocity and pressure data.

The meaning of the metrics shown in Table 4 are provided in Table 3. In the table, *latency_produce* is the time spent on the *simulator* application from the beginning of the simulation to when the simulation data is written to DataSpaces; Metric *latency_consume* is the period from the *Catalyst* application reading the simulation data to the *Catalyst* application generating all visualization data structures. Note that although the second latency is evaluated in the *catalyst* application, it also includes the time taken by the *data processing* application and the *analysis* application, because *Catalyst* always waits for results from those two applications.

From Table 4, we can see that when increasing the number of data processing and simulation processes, the overall end-to-end latency can be significantly reduced. We use a fixed problem size of 4096×4096 and a sample size of 512. Since the divergences generated by these samples are distributed to different *data processing* processes, the divergence calculation time is greatly decreased as we increase the number of *data processing* processes. For cluster ID assignments, since all simulation data is stripped to data processing processes, more processes will result in fewer data regions in each *data processing* process, thus time spent on assigning cluster IDs is also reduced.

7 CONCLUSIONS

This paper presents a flexible and extensible software framework to integrate simulation applications with various analysis applications. A parallel non-parametric clustering method is also designed to perform online machine learning for CFD simulations. In the framework, different applications interact with each other by the means of a shared tuple space. Using DataSpaces can greatly reduce the I/O time compared with the traditional post-hoc approaches. We apply our framework to the application of vortex/anomaly detection in turbulence flows. Using a non-parametric divergence estimation method, a distance-based clustering method is developed to cluster

Table 3: Selected metrics for scalability evaluation.

Metric Name	Description
problem size	case size solved by simulation processes
np_sim	number of processes for simulation application
np_dp	number of processes for data processing application
time_divs_cal	average time spent on divergence calculation in each time step
time_cluster_assign	average time spent on assigning cluster ids using medoids information
time_catalyst	average time used by catalyst application in each timestep
latency_produce	latency produced by the producers: simulation application
latency_consume	latency produced by the consumers: data processing, analysis and visualization
latency_all	overall end-to-end latency

Table 4: configurations and results of scalability evaluation

problem size	np_sim	np_dp	time_divs_cal	time_cluster_assign	time_catalyst	latency_produce	latency_consume	latency_all
4096×4096	16	32	59.3s	88.9s	93.75s	340.5s	453.0s	793.5s
4096×4096	64	128	14.1s	20.1s	91.6s	53.5s	140.2s	193.7s

fluid regions into separate categories. As a result, clustering results and fluid properties such as velocity and pressure are displayed by an in-situ visualization tool at real time. With this new framework, users can get real-time notifications of special patterns or anomalies happening in a turbulence flow.

The paper also presents how different components can efficiently cooperate to achieve a minimized end-to-end latency, and introduces how distributed sampling can provide approximate results with significantly reduced execution time. Furthermore, we design and develop an integrated CFD simulation-time machine learning framework to show how our framework can monitor turbulence flows. The experimental results demonstrate that the integrated framework is efficient and can scale well when more computing resources are used.

ACKNOWLEDGMENTS

This material is based upon research partially supported by the Purdue Research Foundation and by the NSF Grant No. 1522554. Development and experiment of the software framework have used the NSF Extreme Science and Engineering Discovery Environment (XSEDE). Special thanks also go to the DataSpaces team in the Rutgers Discovery Informatics Institute for their assistance during our software development.

REFERENCES

[1] [n. d.]. *Forced isotropic turbulence from Jons Hopkins Turbulence Database(JHTDB).* http://turbulence.pha.jhu.edu/datasets.aspx.
[2] [n. d.]. *John Hopkins Turbulence Database(JHTDB).* http://turbulence.pha.jhu.edu.
[3] [n. d.]. *Kart HPC in Indiana University.* https://kb.iu.edu/d/bezu.
[4] [n. d.]. *OpenFoam.* http://www.openfoam.com/.
[5] Douglas N Ball and HLRS High Performance Computing Center. 2008. *Contributions of CFD to the 787-and Future Needs.*
[6] Jacqueline H Chen, Alok Choudhary, Bronis De Supinski, Matthew DeVries, Evatt R Hawkes, Scott Klasky, Wei-Keng Liao, Kwan-Liu Ma, John Mellor-Crummey, Norbert Podhorszki, et al. 2009. Terascale direct numerical simulations of turbulent combustion using S3D. *Computational Science & Discovery* 2, 1 (2009), 015001.
[7] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G Bruce Berriman, John Good, et al. 2005. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming* 13, 3 (2005), 219–237.
[8] Ciprian Docan, Manish Parashar, and Scott Klasky. 2012. Dataspaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing* 15, 2 (2012), 163–181.
[9] David Ellsworth, Bryan Green, Chris Henze, Patrick Moran, and Timothy Sandstrom. 2006. Concurrent visualization in a production supercomputing environment. *IEEE Transactions on Visualization and Computer Graphics* 12, 5 (2006), 997–1004.
[10] Nathan Fabian, Kenneth Moreland, David Thompson, Andrew C Bauer, Pat Marion, Berk Gevecik, Michel Rasquin, and Kenneth E Jansen. 2011. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on.* IEEE, 89–96.
[11] Evatt R Hawkes, Ramanan Sankaran, James C Sutherland, and Jacqueline H Chen. 2005. Direct numerical simulation of turbulent combustion: fundamental insights towards predictive models. In *Journal of Physics: Conference Series*, Vol. 16. IOP Publishing, 65.
[12] Ming Jiang, Raghu Machiraju, and David Thompson. 2005. Detection and visualization of. *The Visualization Handbook* 295 (2005).
[13] Forrester T Johnson, Edward N Tinoco, and N Jong Yu. 2005. Thirty years of development and application of CFD at Boeing Commercial Airplanes, Seattle. *Computers & Fluids* 34, 10 (2005), 1115–1151.
[14] Leonard Kaufman and Peter J Rousseeuw. 2009. *Finding groups in data: an introduction to cluster analysis.* Vol. 344. John Wiley & Sons.
[15] Jay F Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. 2008. Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments.* ACM, 15–24.
[16] Don O Loftsgaarden, Charles P Quesenberry, et al. 1965. A nonparametric estimate of a multivariate density function. *The Annals of Mathematical Statistics* 36, 3 (1965), 1049–1051.
[17] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A Lee, Jing Tao, and Yang Zhao. 2006. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience* 18, 10 (2006), 1039–1065.
[18] Raymond T. Ng and Jiawei Han. 2002. CLARANS: A method for clustering objects for spatial data mining. *IEEE transactions on knowledge and data engineering* 14, 5 (2002), 1003–1016.
[19] Barnabás Póczos, Liang Xiong, and Jeff Schneider. 2012. Nonparametric divergence estimation with applications to machine learning on distributions. *arXiv preprint arXiv:1202.3758* (2012).
[20] Barnabás Póczos, Liang Xiong, Dougal J Sutherland, and Jeff Schneider. 2012. Support distribution machines. (2012).
[21] John Towns, Timothy Cockerill, Maytal Dahan, Ian Foster, Kelly Gaither, Andrew Grimshaw, Victor Hazlewood, Scott Lathrop, Dave Lifka, Gregory D Peterson, et al. 2014. XSEDE: accelerating scientific discovery. *Computing in Science & Engineering* 16, 5 (2014), 62–74.