X-Composer: Enabling Cross-Environments In-Situ Workflows between HPC and Cloud

Feng Li
Indiana University - Purdue University
Indianapolis, IN, USA
li2251@purdue.edu

Dali Wang
Oak Ridge National Laboratory
Oak Ridge, TN, USA
wangd@ornl.gov

Feng Yan
University of Nevada
Reno, NV, USA
fyan@unr.edu

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

ABSTRACT

As large-scale scientific simulations and big data analyses become more popular, it is increasingly more expensive to store huge amounts of raw simulation results to perform post-analysis. To minimize the expensive data I/O, "in-situ" analysis is a promising approach, where data analysis applications analyze the simulation generated data on the fly without storing it first. However, it is challenging to organize, transform, and transport data at scales between two semantically different ecosystems due to the distinct software and hardware difference. To tackle these challenges, we design and implement the X-Composer framework. X-Composer connects cross-ecosystem applications to form an "in-situ" scientific workflow, and provides a unified approach and recipe for supporting such hybrid in-situ workflows on distributed heterogeneous resources. X-Composer reorganizes simulation data as continuous data streams and feeds them seamlessly into the Cloud-based stream processing services to minimize I/O overheads. For evaluation, we use X-Composer to set up and execute a cross-ecosystem workflow, which consists of a parallel Computational Fluid Dynamics simulation running on HPC, and a distributed Dynamic Mode Decomposition analysis application running on Cloud. Our experimental results show that X-Composer can seamlessly couple HPC and Big Data jobs in their own native environments, achieve good scalability, and provide high-fidelity analytics for ongoing simulations in real-time.

CCS CONCEPTS

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

KEYWORDS

HPC, cloud computing, in-situ data analysis, scientific workflows.

PASC '21, July 5–9, 2021, Geneva, Switzerland

ACM ISBN 978-1-4503-8563-3/21/07.
https://doi.org/10.1145/3468267.3470621

ACM Reference Format:
Feng Li, Dali Wang, Feng Yan, and Fengguang Song. 2021. X-Composer: Enabling Cross-Environments In-Situ Workflows between HPC and Cloud. In *Platform for Advanced Scientific Computing Conference (PASC '21), July 5–9, 2021, Geneva, Switzerland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3468267.3470621

1 INTRODUCTION

HPC and Big Data ecosystems are designed and manufactured for their own purposes (HPC ecosystem for computation-intensive computing and Big Data ecosystem for data-intensive processing and analyses), and hence are significantly different from each other in both software and hardware designs. In the HPC world, software and hardware systems are designed for fastest execution of large-scale parallel computing programs. An HPC system often consists of hundreds of thousands of high-end servers equipped with many CPU cores and large-size memories, which are tightly connected by high-speed interconnects such as InfiniBand. Also, simple and minimal operating systems and software stacks are used in the computer nodes for efficient operations. Moreover, the message-passing library (MPI) is commonly employed in HPC systems such that processes in different address spaces can work collaboratively and exchange data with each other through point-to-point or collective communications.

On the other hand, in the Big Data world, applications are usually designed to collect, process, and analyze large amounts of data to gain important knowledge. For instance, software in a Big Data ecosystem such as Apache Spark [51] and Hadoop [20] can use the high-level MapReduce programming model to execute big data analysis jobs on clusters of commodity machines [12]. More recently, cloud computing technologies such as containers and service-oriented architectures have further hidden the complexity of parallel software packages, and have made Big Data platforms more accessible to developers. Overall, the generic architecture and software design commonly found in Big Data ecosystems can help users quickly analyze data at large scales in an affordable, flexible, and elastic manner. More details of the differences between the HPC and Big Data ecosystems have been discussed by Reed and Dongarra [41].

Although HPC and Big Data ecosystems are different and mostly divergent, in reality, many scientific computing applications have components of both large-scale computations and big data analyses.

For instance, peta-bytes of data may be generated from a single run of a scientific simulation. Also, the simulation results need to be analyzed to get insights or new knowledge. Traditionally, the simulation-generated data is stored in a parallel file system, then copied to another site, and read again by different analysis applications for further investigations. Such a data storage/movement/post-analysis pattern is often extremely expensive, and hence there is an inevitable trend to pursue in-situ data analysis. In a typical in-situ data analysis, the analysis applications continuously process/analyze the in-memory data structures while the simulation applications are running and generating data [6, 16, 18, 23].

Most existing in-situ computing frameworks are designed for executing in the HPC ecosystem (in many cases, involving one or two HPC systems). To support HPC-based in-situ computing, any required data analysis applications must be ported and pre-compiled for the HPC systems. If the data analysis application needs special data-processing functionalities such as aggregation, group operation, they have to be manually reimplemented using the common parallel programming models such as MPI and OpenMP. However, cloud computing platforms already provide mature and easy-to-use tools and environments for data processing and machine learning based analysis. For example, the Helios system from Microsoft [40] is designed for large scale data ingestion, indexing, and efficient queries. In Helios, telemetry data (e.g., logs, heartbeat information) are formatted as easy-to-understand data records, and a variety of tasks such as monitoring, debugging, and reporting are automatically supported.

In the HPC world, however, complex data processing and analytics systems like Helis are mostly absent or hard to deploy, and it is important yet challenging to utilize Big Data toolchains to analyze HPC simulations on HPC systems. Porting Big Data toolchains to HPC systems can be a temporal solution [33, 34]. However, with the bigger demands in using Big Data toolchains, the fast growing of Big Data ecosystem, and the increasingly more complex software-hardware co-design optimizations, such a temporal solution requires tremendous manual efforts that are not efficient nor scalable. A more promising alternative approach is to have simulation data transformed into the formats supported by Big Data toolchains (e.g., indexed records) in an automated and efficient manner and then stream them to external services deployed in Cloud platforms on the fly. With simulation data formatted as indexed records, Cloud-based online stream processing systems can utilize the data layout semantics to conduct large-scale complex analysis. For example, Dynamic Mode Decomposition (DMD) analysis in Section 2.3 requires data aggregation in both time and space, which fits very well in a typical stream processing scenario. Section 4 analyzes the effect of data communication overhead between HPC and Cloud systems.

To this end, we design and implement the X-Composer software framework, which seamlessly bridges the separate ecosystems of HPC and Cloud. To the best of our knowledge, this work is the first to compose in-situ workflows that consist of both HPC-native and Cloud-native application components, and enable transparent yet efficient data transformation between the two semantically different ecosystems. Specifically, we regard simulation data as records that can be indexed by its fields, so that complex operations such as filtering, reduction, aggregation can be realized using high-level abstractions. By bridging HPC simulations with external Cloud-based analytics, the elastic services and native software in the Cloud can be utilized to analyze HPC-based simulations effectively without lots of re-implementation time and efforts. Moreover, the elasticity of Cloud can help our data analytic services scale up and out easily when there is more data to be processed.

X-Composer includes a client library and a deployable manager component. MPI applications can be linked with the X-Composer client library, and invoke corresponding library calls to generate indexed data. The manager component (X-Composer Manager) supports authentication of Cloud/HPC systems, and can submit jobs to both sides using their native schedulers (e.g. Slurm [50] for HPC and Kubernetes [7] for Cloud). X-Composer Manager launches cross-environment workflow using two-level scheduling: at the first level, X-Composer Manager utilizes the platform-specific authorization services to deploy/configure applications to the computing platforms; at the second level, HPC batch jobs and Cloud analysis tasks are launched with the native schedulers. When X-Composer submits jobs to HPC, the simulation data is transformed to Cloud-native indexed records, and continuously streamed to the data analysis services deployed in Cloud, where the data objects together with their scheme information are organized and analyzed. The X-Composer framework provides several essential features to realize cross-environments in-situ workflows. Firstly, X-Composer lets users launch and execute such cross-environment workflows in a secure and protected way, by utilizing the native authorization services in both platforms. Secondly, X-Composer handles necessary data transformations between Cloud and HPC platforms automatically, so that simulation applications only need to be modified slightly to enable rich analytical services provided by external Cloud platforms. Furthermore, X-Composer provides flexible configurations for cross-environment workflows. For example, the mapping between simulation processes and data analysis endpoints can be customized to minimize data transmission overhead and minimize data flow stalls.

To evaluate the X-Composer system, we develop a real-world cross-ecosystem scientific workflow, which has a parallel MPI-based computational fluid dynamics (CFD) simulation running in HPC, and a distributed online DMD analysis application using stream processing service deployed in Cloud. We build and execute this workflow on the TACC Stampede2 HPC [46] and XSEDE Jetstream Cloud systems [45, 48]. From the experimental results, we observe that by linking CFD applications with X-Composer, we can effectively migrate the simulation data from HPC system. Compared with the traditional file-system based approach, X-Composer reduces the elapsed time of the workflow by 1.5 times, while using 512 MPI processes and 8 Cloud endpoints. Also, by using the remote Cloud analysis services, we are able to provide online timely insights into the ongoing fluid dynamics.

In the rest of the paper, the next section introduces the background of in-situ processing on HPC and stream processing on Cloud. We present the system design of the proposed X-Composer framework in Section 3. Then we demonstrate the usage of X-Composer with real-world in-situ workflows in Section 4, and show its performance. In Section 5 we discuss recent related work on in-situ processing and workflow management. In Section 6, we conclude the paper with future work.

2 BACKGROUND

In this section, we first introduce HPC in-situ processing and Cloud-based stream processing. Then, we introduce the Dynamic Mode Decomposition, which is a data analysis method we have deployed in our Cloud-based stream processing service.

2.1 In-situ processing on HPC systems

Originally, the Latin world term `in situ` has the meaning of "on-site" or "in place". Researchers have been using "in-situ" to describe the situation when visualization/analysis programs can process simulation data without any data movements or if the visualization/analysis routines reside in the same processes with simulation [6, 44]. There is also a broader definition of "in-situ" processing: processing data while it is generated [9, 11, 23, 32]. More detailed terminology description and classification of in-situ has been introduced [10]. In this paper, we use the broader definition of in-situ, for which parallel simulations are running in an HPC system, at the same time data is continuously offloaded to a Cloud system for further data analysis.

Generally, in-situ analysis can be realized in different ways on HPC systems. In a *tightly-coupled* model, parallel applications (e.g., LAMMPS [39]) let users couple simulation with other code either by linking LAMMPS as a library or letting LAMMPS call the other code. In a *loosely-coupled* model (e.g., when using DataSpaces [16], ADIOS2 [24], or Decaf [18] libraries), both simulation and analysis applications need to call specific transport library interfaces, so that data movement can be carried out by the specific data transport runtime system. Also, both loosely-coupled and tightly-coupled models require modifying simulation and analysis applications' code at a low level, so that it can be linked with the corresponding transport libraries.

Cloud platforms typically provide more robust and easy-to-use programming abstractions. For example, a "word count" application on a data stream can be simply implemented in Spark Streaming in one line as: `input.as[String].flatMap(_.split("␣")).groupBy("value").count()`. This application reads the incoming data records, splits them into separate words, and counts the occurrences of each word. Under the hood, Spark runtime automatically partitions the data records and uses multiple executors to "count" in parallel. Such high-level operations make developers handle parallelism easily, without worrying about the low-level programming primitives (e.g. MPI) or manually dealing with concurrent access to shared variables. We argue that it's easier and more efficient for analysis applications to implement application logic with such high-level data abstractions, which are usually provided by modern distributed stream processing frameworks.

2.2 Cloud-based stream processing data analytics

Nowadays, it has become common that data is generated continuously over time. For example, sensor data generated from IoT devices or web logs are produced from multiple sources and can accumulate quickly every day. Instead of storing the data and doing post-processing in the future, stream processing can be used to give real-time insights into the data. The advantage of being "real-time" is essential in various scenarios such as online fraud detection and emergency handling, where it can help early decision-making.

In data stream processing, "unbounded" datasets (or "data streams") are used as input. New data records are continuously added to the data streams, where they can be analyzed on the fly. Popular stream processing frameworks (e.g., Apache Kafka [30], Flink [8], Storm[21], and Spark Streaming [22]) have been extensively used in different domain fields to provide in-time analytics for various data sources. Popular Cloud providers now also offer big data analytics as a service (e.g., Google DataProc [27], Amazon Kinesis Streams [26]), so that users can interact with the service using their favorite programming languages (e.g. Python), regardless of platform infrastructure. In the case of computational fluid dynamics (CFD) in the HPC domain, scientific simulations can run over days or even months, and real-time analysis of simulation results can significantly help scientists analyze ongoing simulations. For instance, analysis of data generated while the simulation is in progress can help scientists discover important patterns and understand behaviors, which users would otherwise have to wait till the simulation finishes. In this work, we explicitly utilize the convenience and advantages of Cloud-based stream processing to provide timely insights into the running scientific simulations.

2.3 Analytical Method of Dynamic Mode Decomposition

In fluid dynamics, the flow fields are organized in a complex, high-dimensional dynamical system. It is well known that important flow features can be recognized through visual inspections of the flow, even when there are perturbations and variations [47]. This means that some coherent structures exist in the fluid fields, which contain useful dynamical information of the fluids and can help researchers understand the patterns/behaviors of the fluid flows. To mathematically extract those coherent structures from such dynamical systems, modal analysis techniques, such as Dynamic Mode Decomposition analysis (DMD [43]), are often used. More specifically, the DMD data analysis solely relies on snapshots (or measurements) of a given dynamic system, and provides the spatial-temporal decomposition of those data into a set of dynamical modes [31]. Since DMD is data-driven and does not need to model the governing equations of the fluids, DMD is considered as an "equation-free" and "data-driven" method. Traditionally, DMD analysis has been used to study fluid structures from dynamic flow geometries [42]. In this paper, we use DMD as an analysis example, and show how it can be deployed in the Cloud as a part of the distributed stream processing service, to analyze ongoing CFD simulations at real-time.

3 METHODOLOGY

In this section, we present the design of X-Composer and explain how X-Composer solves the challenges of offloading analytical tasks to Cloud systems from the running simulations. A high-level design overview of X-Composer is present in Figure 1.

Typically, numerical simulations such as CFD or MD simulations are computation intensive applications. In HPC, "scheduling" is done by batch schedulers such as Slum. In a Slurm job on HPC, users describe the number/type of nodes they require, and how they want

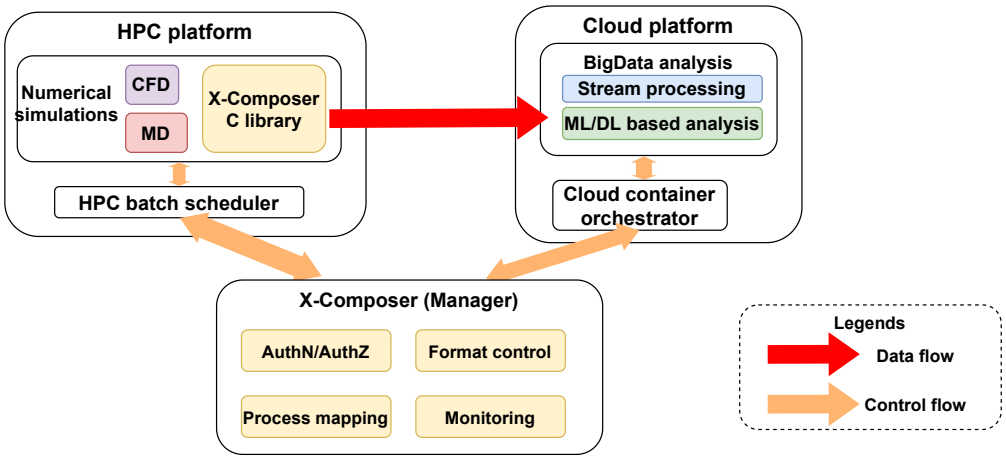

Figure 1: High-level design of X-Composer. X-Composer provides a C client library which HPC applications (e.g. CFD or MD simulations) can link with. X-Composer Manager can launch cross-environment workflows using HPC and Cloud credentials. It also configures the format conversion, process/dataflow mapping between HPC and Cloud applications. Native HPC scheduler and Cloud orchestrator are used by X-Composer to submit tasks to both Cloud and HPC environments.

to arrange the MPI processes on compute nodes for efficient execution. On the right-hand side, BigData analysis is typically deployed in Cloud platforms. Analysis tasks such as data stream processing, machine learning (ML) or deep learning (DL) based analysis are usually packaged as containers. Container orchestration tools such as Kubernetes make it possible to scale out the analysis processes for increasingly large amounts of data.

The X-Composer Manager sits between HPC and Cloud platforms. It manages the authentication/authorization of both sides, and prepares job submission to the native job schedulers in both platforms (as seen in the control flows in Figure 1). The X-Composer Manager first launches the Cloud services and HPC simulations respectively using the credentials provided by the user. A shared token is copied to both sides so that the client library can securely connect to the Cloud endpoints. After the cross-environment workflow is established, data flow happens solely between HPC and Cloud components, without passing through the X-Composer Manager. To easily patch traditional parallel simulations with Big Data Analytics capabilities, X-Composer provides the following features:

- Provides a C library that parallel applications can be linked with, so that they can transform the in-data data structures into indexed records on the fly.
- Coordinates job submissions to both HPC and Cloud platforms, and manages authentication in both platforms.
- Provides users with a simple interface to define various types of interactions between HPC processes and Cloud endpoints.

In the rest of this section, we will present more details about how we design X-Composer. We first introduce the HPC and Cloud components, and then show how X-Composer bridges the two ecosystems using the HPC and Cloud native schedulers to enable cross-platform in-situ workflows.

3.1 HPC components

The HPC environment has a simpler software stack than the Cloud environment for efficient parallel executions. For this reason, we provide an HPC-side client library, which has a simple interface

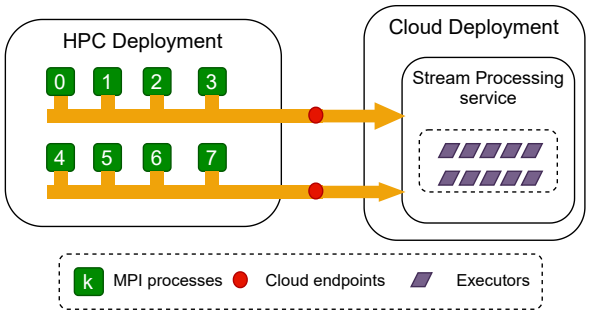

Figure 2: Binding MPI process to Cloud endpoints. In this example, processes 0 – 3, 4 – 7 are connected to two different Cloud endpoints.

for HPC applications so that it allows existing simulation code to easily adapt to the X-Composer services. During the runtime of the simulation applications, in-memory data structures are converted into indexed data records, and then sent out to Cloud while the simulation is running. For example, an example record can contain such information: `process_id, time_step, pressure`, where the `pressure` field includes the temperature information of all fluids simulated by a process at a certain time step.

The C library interface provided by X-Composer is shown in Listing 1. In the defined API, the `xcomposer_init` function initializes the connections between HPC and Cloud by registering data fields from the simulation to the remote Cloud service endpoints. The data fields are denoted by the *field_name* variable in the API, such as "pressure" or "velocity_x". The `xcomposer_init` function also registers each MPI process rank to one of the existing Cloud endpoints for future data writes. To output multiple fields simultaneously, the initialization function can be called several times with different instances of `field_name`.

During the time-step based scientific simulations, the `xcomposer_put` function is called iteratively, to transform the field data output from the simulation processes into stream records, which are sent to the Cloud endpoint the process has connected to. Each stream record contains the time-step information and the serialized field data of

the simulation process. On the Cloud side, stream records received from all endpoints will be aggregated, and analyzed by the stream processing service, which will be discussed in Section 3.2.

```
// initialize the X-Composer service, by connecting each MPI
// process with one of the Cloud endpoints.
xcomposer_ctx* xcomposer_init(const char *field_name,
                MPI_Comm comm, char *endpoint_file_path);

// write a chunk of in-memory data (void* values) out.
int xcomposer_put(xcomposer_ctx *context, int stepid,
                void* values, size_t data_len);

// finalize the X-Composer services.
int xcomposer_finalize(xcomposer_ctx* context);
```

Listing 1: The X-Composer C client API.

To support different types of mappings between MPI processes and Cloud endpoints, X-Composer provides a utility Python API. In X-Composer, the default mapping scheme, called "contiguous mapping", evenly divides all MPI ranks into contiguous groups, and assigns each group one Cloud endpoint, as shown in Figure 2. Assigning separate Cloud endpoints for different groups of processes enables us to achieve a higher data transfer rate. Process grouping also provides a higher degree of flexibility. Users can decide how many endpoints are necessary based on the outbound bandwidth of HPC sites and inbound bandwidth of each Cloud endpoint, as shown in Listing 2.

```
def contiguous_mapper(mpi_rank, mpi_size, num_endpoints):
    group_size = math.ceil(mpi_size/num_endpoints) #round up.
    local_id = mpi_rank % group_size
    group_id = mpi_rank // group_size
    return (group_id, local_id)
```

Listing 2: An example of defining contiguous mapping between MPI processes and Cloud endpoints using the provided Python API. Users can specify different patterns of mapping by providing similar Python scripts to the X-Composer Manager.

3.2 Cloud-based data analysis components

In this subsection, we first introduce how we set up the Cloud stream processing service, and then describe how different components in the Cloud service are bound together to analyze the incoming streamed simulation data to provide insights.

We choose `Spark Streaming` [22] as our example stream processing engine, which supports scalable, high-throughput, fault-tolerant stream processing of live data streams. We choose Spark Streaming over other streaming engines (e.g., Apache Storm/Heron) because the Spark ecosystem is more widely used, and has better analysis library support. With the core Spark functionality, we are able to apply standard data operations to streams such as map, reduce, filter, join, as well as realizing advanced algorithms by directly calling Spark Machine Learning and Graph Processing libraries. In this paper, we deploy a Spark cluster and Cloud endpoints within a Kubernetes cluster in the Jetstream Cloud [45]. As a popular container orchestration system, Kubernetes provides an abstraction layer above different Cloud providers [7]. This way our stream processing component can be easily reproduced on different Cloud providers like Google Cloud Platform or Amazon AWS.

Figure 3 shows the overall design of our Cloud setting with the Spark stream processing engine. In the current X-Composer prototype implementation, we use Redis server instances as our Cloud endpoints. Redis[1] is an in-memory data structure store, and used to accept data streams from the HPC components. We use `spark-redis` connector[2] to let the Redis instances forward structured data to Spark stream processing services deployed in Cloud. Note that X-Composer is designed as a general-purpose middleware to enable cross-platform in-situ workflows. Thus the Cloud analysis component is not limited to Spark, and other programming environments and models can also be supported by X-Composer.

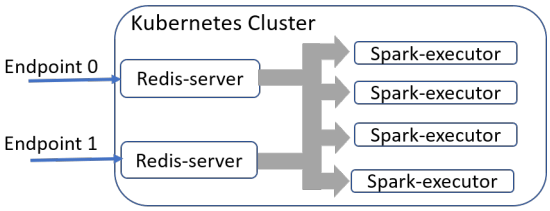

Figure 3: The deployment of our Cloud components. Each Redis-server container acts as an endpoint, and exposes the same TCP port to outside. The Spark-executor containers will read available data streams from Redis-server containers. All containers are scheduled in the Kubernetes cluster deployed in Jetstream, and use the in-cluster network to communicate with each other.

All the Redis instances export TCP port 6379 to the outside of the Cloud system. All of our Cloud services (Spark stream processing engine and Redis server instances) are containerized and are scheduled using Kubernetes's native scheduling, which makes users' work much easier to adapt to different Cloud providers. More specifically, our Spark-executor container built in X-Composer is comprised of the Python-based DMD analysis library PyDMD [15], and several Scala software packages such as spark-redis. More details about the software we use in the Cloud services of X-Composer are described in Section 4. Also, we create and manage the Kubernetes cluster in the Jetstream Cloud by using a customized Kubespray software [3]. After the Kubernetes cluster is set up, we create a Kubernetes service account and store its credentials in the X-Composer Manager. With the locally stored credentials, X-Composer Manager can connect to the Kubernetes cluster and launch the pre-built containers remotely.

3.2.1 Data Processing in Cloud. When data is aggregated from different endpoints, Spark-executors will read records from data streams sent by all MPI processes. Fluid data (in terms of snapshots) from different simulation processes are added to the separate data streams over time.

Figure 4 shows how data records in one data stream are aggregated as Spark "Dataframes", which are then processed by analysis code. In X-Composer, we let Spark manage the task scheduling and parallelism, so that multiple executors can be mapped to different data streams and process the incoming data concurrently. Each data stream will be mapped to a partition of RDD. We use the `rdd.pipe` function in Spark to send Dataframe data from the main Spark context to external programs (in our case the Python interpreter). This operation happens concurrently with all data streams, thanks to the design of Spark that enables a high degree of parallelism. The results

[1] https://redis.io
[2] https://github.com/RedisLabs/spark-redis
[3] https://github.com/zonca/jetstream_kubespray

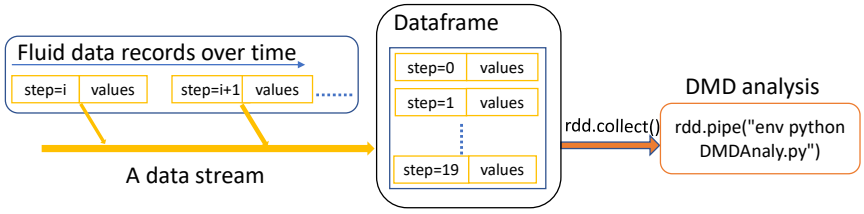

Figure 4: Data processing in the Cloud. Each MPI process sends data through a data stream, then unbounded data in each data stream is re-arranged into micro-batches (aka Spark Dataframes). Micro-batches from multiple data streams are treated as partitions of one Resilient Distributed Dataset (RDD). The `rdd.pipe` function then sends each partition to the external Python script exactly once.

of all Spark-executors are then collected using the `rdd.collect` function so that they can be visualized/presented.

3.3 Support for Authentication and Security

Since there are separate X-Composer components running across geographically distributed computing platforms, it is important that both control flow and data flow are protected. Here we discuss how we support the authentication/security in X-Composer.

During the workflow setup, X-Composer can access both HPC and Cloud platforms securely. Currently we deploy our X-Composer Manager in the HPC login node, which has a secure access to HPC computing resources by default. For the Cloud platform, we assume there is a ready-to-use Kubernetes cluster (either managed by the user directly or orchestrated by Cloud providers like Google Cloud/AWS). X-Composer Manager stores the Kubernetes service account credentials, so that it can create/control/delete containers remotely. A random token is also generated by X-Composer Manager in this setup stage, which will be used in later workflow executions.

During the workflow execution, the random token generated from the setup stage is used to protect the data flow between HPC and Cloud systems. Specifically, all Redis instances in the Cloud use this token as password to accept incoming connections. This token is also composed in HPC Slurm jobs so that HPC applications can initiate secure dataflow to Cloud endpoints.

Connecting HPC to outside clouds: In a typical HPC system, only the login nodes and designated gateway nodes are accessible to the public network. Incoming connections to compute nodes are not allowed for security reasons. However, most HPC systems allow outbound traffic from compute nodes to the outside. In systems like Stampede2, such outbound connections are allowed by default; in other systems such as PSC Bridges, users are can specific SLURM options (e.g. "-egress") to enable outbound traffic from compute nodes. Such outbound traffic is usually routed to one or several gateway nodes and then travels to public network. In X-Composer, we launch the Cloud analysis and expose the endpoints first, so that processes in HPC can initiate connections to them.

4 EXPERIMENTAL RESULTS

We perform two sets of experiments to evaluate the performance of in-situ scientific workflows using X-Composer. The first set of experiments use a real-world CFD simulation running in HPC, with DMD analysis running in Cloud, to show the workflow's good end-to-end time. The second set of experiments use synthetic workflows across HPC/Cloud to evaluate X-Composer's throughput and quality of service at different scales.

We use TACC Stampede2 as our HPC platform to run CFD simulations, and use TACC Jetstream Cloud [45, 48] as our Cloud platform to run DMD data analysis. The specifications of Stampede2 and Jetstream systems are shown in Table 1a and 1b, respectively.

4.1 Design and implementation of a hybrid HPC-Cloud CFD scientific workflow

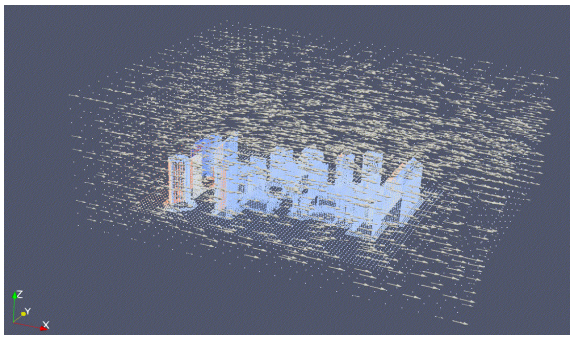

Figure 5: Visualization of *WindAroundBuildings* simulation using ParaView [1]. In this figure, arrows show the velocity of the wind, and colors in the buildings represent different pressure.

Our cross-environment in-situ scientific workflow consists of two applications: CFD simulation and DMD analysis. For the CFD simulation application, we use the parallel OpenFOAM software [28, 38] to implement it, and execute it on TACC Stampede2. In OpenFOAM, a "solver" refers to a simulation algorithm (e.g., using LES [13] or DNS [36]); and a "case" describes the physical condition of the simulation problem. We choose `simpleFoam` as our solver, which is a steady-state solver for incompressible, turbulent flow, using the SIMPLE (Semi-Implicit Method for Pressure Linked Equations) algorithm.

The simulation problem we solve is the `WindAroundBuildings` problem, which is displayed in Figure 5. This specific computation can simulate how wind flows will behave around a group of buildings in an urban area. To support such an in-situ workflow execution with X-Composer, we replace the original `runTime().write` function in the `simpleFoam` solver with our `xcomposer_put` function, and link the solver executable with our X-Composer client library. We run the simulation problem with a process layout of $1 \times 1 \times 16$ (in x, y, z directions respectively), so that each process is responsible for a horizontal slice of the whole problem space. The

Table 1: Hardware and software information of the TACC Stampede2 HPC system and the XSEDE Jetstream Cloud system. Jetstream and Stampede2 are connected through 40 x 4 Gbps network in TACC.

(a) Stampde2 HPC

	Information
CPU	Intel Xeon Phi 7250
#cores	68 cores per node
Main memory	96 GB DDR4 per node
Network	100 Gb/s Intel Omni-Path
MPI	Intel IMPI 18.02
Compiler version	GCC 5.4
OpenFOAM	v7
Lustre filesystem	30 PB, 4 MDSs & 66 OSTs

(b) Jetstream Cloud

	Information
Host CPU	24 cores
Host RAM	128 GB
Host storage	2 TB
Host network	10 gigabit Ethernet
VM image	Ubuntu-1804-latest
VM type	m1.large
VM vCPUs	10
VM memory	30 GB
Kubernetes version	1.17.6
Spark version	3.0.1
Redis version	5.0

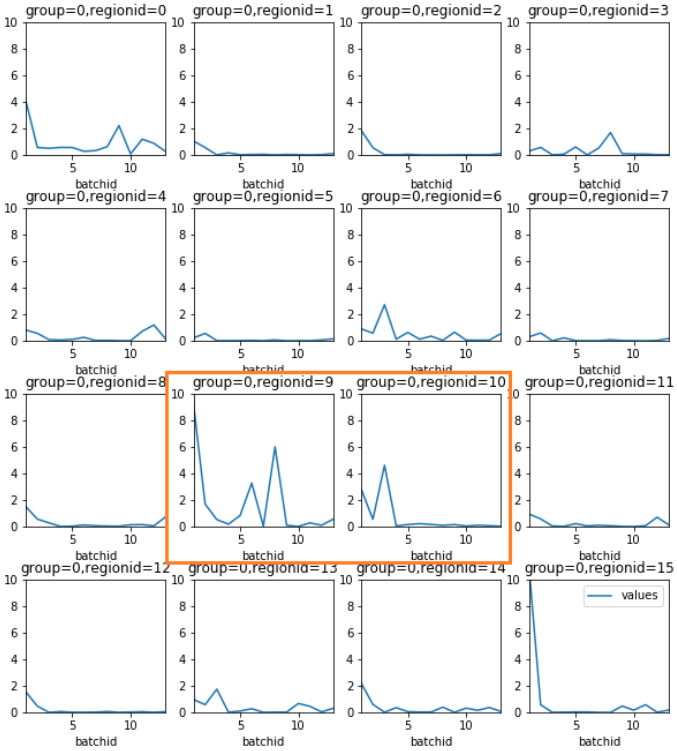

Figure 6: Analysis results of eigenvalues of DMD low-rank operator from each process region's output. Each subplot shows the average sum of square distances from eigenvalues to the unit circle of that region. Values closer to 0 mean fluids in that region are more stable.

velocity fields of each process region are streamed to Cloud service providers through the X-Composer client library, and are continuously analyzed by the DMD service deployed in the Jetstream Cloud.

On the Cloud side, the data analysis application (or the DMD service) reads streams sent from HPC processes through the Cloud endpoints described in Section 3.1. Figure 6 shows the visualization results of the DMD analysis on 16 data streams received by 1 Cloud endpoint. Within the figure, each subplot shows the DMD result of the fluid data from one CFD simulation process. Each subplot reveals how the fluid dynamics change over time for a particular domain region. This figure can also inform users how stable each region's fluid flow is at real-time, while the simulation is still running. For instance, regions 9 and 10 in the figure (marked in orange color) shows high eigenvalues, and can easily tell users that those regions are significantly unstable and may need further analysis.

4.2 End-to-end workflow time

One issue of in-situ processing is that the data analysis application may slow down the simulation application, increasing the overall end-to-end time of the workflow. Traditionally, simulation applications write output to a parallel file system. Later the stored data files are read for post-analysis. Such file-based I/O is usually expensive, and can severely slow down the primary simulation applications.

To investigate how the simulation application and the combined in-situ workflow (with Cloud-based DMD analysis) can be affected by different I/O methods, we configure the simpleFoam solver (with 512 processes) in three different modes:

(1) File system-based: simulation output data is written to a parallel Lustre file system using the "collated" write method provided in OpenFOAM.
(2) X-Composer: simulation output data is sent to Cloud endpoints, using the proposed X-Composer API.
(3) Simulation-only: The simulation runs with all data writes disabled.

We use a slightly modified WindAroundBuildings case in our experiments, which has finer cells (200, 160, 80 cells in the x, y, z directions, respectively) so that the simulation can utilize more processes. The simulation is configured to run 1000 steps, and `writeInterval` is set as 5 (steps). The simulation is launched with 512 processes (on 8 Stampede2 KNL nodes) with a process layout of $8 \times 8 \times 8$ (in x, y, z directions respectively), using the "hierarchical" decomposition method in OpenFOAM. With such a configuration, $47.8MB$ data are generated by all processes in each step. Hence, the total 1000 steps will generate $47.8MB * 1000/5 = 9.56GB$ of data, where 5 is the `writeInterval`.

Figure 7 shows the time breakdown when we run our WindAroundBuildings case in file system-based, X-Composer and simulation only modes, respectively. For the file system-based mode, we use the Lustre system described at Table 1a. For the X-Composer mode, we use the contiguous mapping with 8 Cloud endpoints. From the figure, we can see that the file system-based mode takes

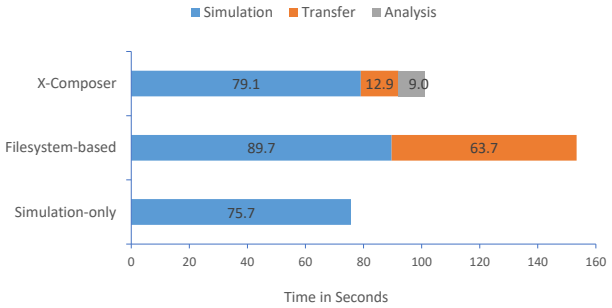

Figure 7: Simulation elapsed time comparison when running *WindAround-Buildings* case with file-based I/O, X-Composer and simulation-only. The figure shows that while file-based I/O significantly slows down the simulation application, the proposed X-Composer method doesn't affect simulation much.

significantly more time compared with the simulation-only mode. The main reason is that the "collated" file write operation requires coordination of simulation processes which stalls the simulation processes. Note that in the "collated" I/O method[4] provided by OpenFOAM, the outputs from all MPI processes are collated to a single file that is then written by the master processor.

In comparison, when the X-Composer mode is used, simulation applications can run with a minimal slowdown. This is due to the fact that X-Composer asynchronously writes in-process simulation data to streams, from each simulation process, independently. Compared with the file-based method, no shared file systems are used for the output of the bulk simulation, so the simulation can run with much fewer stalls. From this experiment, we see that integrating CFD simulations with X-Composer can give users timely insights into the ongoing simulations, meanwhile not significantly slowing down the performance of the simulations.

4.3 Throughput

In order to better understand the throughput performance of our proposed X-Composer system, we conduct the second set of experiments, in which we demonstrate how the system will scale when we use more HPC and Cloud resources.

In the second set of experiments, we use a synthetic data generator to produce output data at different rates in order to stress the system. The synthetic data generator is an MPI application where each process generates simulation data at a user-specified rate. Data generated from all processes are streamed to the distributed stream processing service through multiple Cloud endpoints, as we have seen in Subsection 3.1.

Figure 8 shows the aggregated throughput from all data generator processes with the process number ranging from 64 to 512. Each data generator process generates data at a rate of 0.68MB/s. As the number of data generator processes increases, the required total communication volume will increase linearly. In our experiments, when we use a larger number of process numbers, we also need to increase the number of Spark-executors and Cloud endpoints (i.e., Redis server instances) correspondingly. The ratio among MPI processes, Cloud endpoints, and Spark-executors is set as $64:1:16$.

[4]The default "uncollated" I/O method in OpenFOAM gives worse performance due to frequent metadata operations applied to a large number of files.

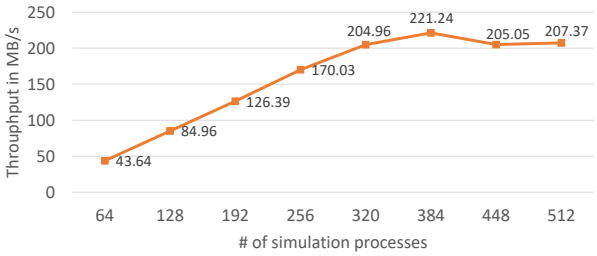

Figure 8: Running workflow with a synthetic data generator in HPC, and Spark-based query analysis in Cloud. Each group of 64 MPI processes write data to the same Cloud endpoint, which is served by 16 spark executors in Cloud.

From Figure 8, we can observe that when we run the synthetic simulation experiments with less than 320 processes, the in-situ workflow throughput increases linearly with the number of processes. However, when there are more than 320 processes, the throughput does not increase anymore and it plateaus at 221.24 MB/s. We have also used the iperf benchmark tool [25] to measure the bandwidth between TACC Stampede2 and Jetstream, which reports around 450 MB/s aggregated bandwidth when we use 512 MPI processes and 8 Cloud endpoints. We acknowledge that our highest throughput of 221.24 MB/s is less than the physical network bandwidth between the TACC Stampede2 and Jetstream systems. To understand the performance issue, we have also measured the throughput between Stampede2/Jetstream environments by just using the Redis built-in benchmark library [5], and we found out the resulting throughput is also significantly lower than the raw bandwidth measured by the iperf tools. The parallel *hiredis* library — that provides the Redis API — uses the request-reply communication model, which requires packets to travel from the client to the server, then back from the server to the client to carry the reply. Although we have already applied general optimization techniques such as pipelining, the current implementation still cannot utilize the full link speed between TACC Stampede2 and Jetstream Cloud. We expect that using the other asynchronous messaging libraries such as ZeroMQ [53] may improve the throughput further, and we plan to investigate and add support for them in the future.

5 RELATED WORK

Scientific workflows have been widely used to incorporate multiple decoupled applications running on distributed computational resources. To manage data dependencies among different applications, and correctly schedule computational tasks, workflow management systems (e.g., Pegasus [14], Kepler [35]) are often used. However, these workflow systems heavily rely on file-based I/O, and only schedule coarse-grain workflow tasks in a sequential manner (i.e., a later task cannot start until all the previous tasks have exited). In the case of our X-Composer, simulation data is streamed continuously to Cloud services, where data analysis will be conducted while the simulations continue running.

There exist several previous works to enable advanced analysis on scientific data through customized analysis operations. For instance, LABIOS [29] utilizes the label-based I/O system to bridge

[5]https://redis.io/topics/benchmarks

HPC and Big Data applications. NIOBE [19] uses I/O forwarding nodes and burst buffer to stage data and offload the data format conversion operations. ArrayUDF [17] provides a user-defined functions (UDF) abstraction for multi-dimensional arrays with generalized structural locality support, so that computation that requires neighborhood information can be carried out with the high-level UDF interface. However, these works typically require a shared file system or shared storage system.

Data transport libraries such as ADIOS [32], Decaf [18], and Zipper [23] can be used to construct in-situ workflows that consist of separate parallel applications. Paraview Catalyst [2] and SENSEI [3] utilize the standard visualization toolkit (VTK) data representation, and allow simulation and visualization/analysis applications coupled through customized "adapters". Although those solutions do not rely on file-based communications between applications, they most often require applications to run in an HPC ecosystem. Differently, in X-Composer, data can be sent from HPC applications to endpoints exposed by Cloud services, so that decoupled applications can collaborate while residing in their native environments.

There are a few recent works targeting constructing workflows in heterogeneous environments. Parsl [4] is a library that allows developers to express parallelism of workflows in Python. It uses Python objects, files, and *futures* as the input of applications. DagOn* [37] is a Python-based workflow engine that allows users to define parallel jobs represented by directed acyclic graphs on a combination of local machines, remote servers, and cloud-based infrastructures. Beeflow [9] is a workflow management system that supports traditional workflows as well as workflows with in situ analysis, and it utilizes events-synchronization primitives to enforce in situ workflow logic. Yildiz et al. [49] explore the combination of task-based computing model and in situ workflows using Decaf and PyCOMPs, by integrating Decaf's in situ components as a sub-workflow of the task-based PyCOMPs workflow. Badia et al. [5] use a holistic way to express a workflow, where both data/computing are integrated into a single flow built on high-level interfaces. Most of these works still use file abstractions as the dependencies between tasks, and the consumer of the tasks cannot utilize the semantics of the data. In comparison, X-Composer treats data as continuous streams of indexed records, and supports a single in-situ workflow running across distinct HPC and Big Data ecosystems, thus high-level data operations such as filtering, grouping can be easily supported.

Zanuz et al. [52] proposed a framework to run an in-transit workflow consisting of parallel MD simulation and distributed stream processing using Flink. In their setup, all BigData tools are deployed in the same cluster where the parallel MPI-based simulation runs. From our experience, setting up BigData tool-chains for HPC environments and can require significant administrative assistance. In comparison, X-Composer explicitly places the Cloud-friendly analysis (such as stream processing) to the external Cloud platform, and utilizes Cloud technologies such as container orchestration to make the deployment and management easier.

6 CONCLUSION AND FUTURE WORK

In this paper, we design a prototype in-situ workflow framework X-Composer to bridge the HPC and Cloud ecosystems. X-Composer automates the construction and management of the complex interactions and dataflows between the native applications deployed in HPC and Cloud. X-Composer also provides a unified approach and recipe for supporting hybrid in-situ workflows on distributed heterogeneous resources. X-Composer transforms simulation data generated in the HPC into stream records, and sends the stream records to a distributed stream processing service deployed in Cloud. Through a case study using real applications, we demonstrate how the Cloud-based stream processing service is set up, and how it partitions, processes, and analyzes the stream data continuously. We use the parallel OpenFOAM simulation which runs on TACC Stampede2, and DMD analysis which is deployed in the XSEDE Jetstream Cloud to verify the effectiveness of our framework. Experimental results show that extending MPI-based simulations via X-Composer enables stream processing services deployed in Cloud for providing in-time analysis of ongoing fluid dynamics. The experiments also show good throughput and quality of service of X-Composer at moderate scales.

In our future work, we plan to extend X-Composer to support in-situ workflows depicted in more complex directed acyclic graphs (DAG). More advanced monitoring and failure recovery mechanisms can be added to X-Composer to provide support for complex workflows. More advanced data aggregation functionality can be supported on the HPC side so that processes may utilize the bandwidth more efficiently, Additionally, analytical performance models can be designed to automatically decide how to distribute computation tasks of an in-situ workflow to different environments (e.g., HPC and Cloud), based upon application-specific requirements such as computation time, waiting time in queues, memory consumption, and migration cost.

ACKNOWLEDGMENTS

This research is supported by the NSF award #1835817 and the DOE contract #DE-AC05-00OR22725. This work also used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by NSF grant number ACI-1548562.

REFERENCES

[1] J. Ahrens, Berk Geveci, and C. Law. 2005. ParaView: An End-User Tool for Large-Data Visualization. In *The Visualization Handbook*. Elsevier.

[2] Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O'Leary, Kenneth Moreland, Nathan Fabian, and Jeffrey Mauldin. 2015. ParaView Catalyst: Enabling In Situ Data Analysis and Visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV2015)*. ACM, New York, NY, USA, 25–29.

[3] Utkarsh Ayachit, Brad Whitlock, Matthew Wolf, Burlen Loring, Berk Geveci, David Lonie, and E. Wes Bethel. 2016. The SENSEI Generic In Situ Interface. In *2016 Second Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*. 40–44.

[4] Yadu Babuji, Anna Woodard, Zhuozhao Li, Daniel S Katz, Ben Clifford, Rohan Kumar, Lukasz Lacinski, Ryan Chard, Justin M Wozniak, Ian Foster, et al. 2019. Parsl: Pervasive parallel programming in python. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*. 25–36.

[5] Rosa M Badia, Jorge Ejarque, Francesc Lordan, Daniele Lezzi, Javier Conejero, Javier Álvarez Cid-Fuentes, Yolanda Becerra, and Anna Queralt. 2019. Workflow environments for advanced cyberinfrastructure platforms. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1720–1729.

[6] Janine C Bennett, Hasan Abbasi, Peer-Timo Bremer, Ray Grout, Attila Gyulassy, Tong Jin, Scott Klasky, Hemanth Kolla, Manish Parashar, Valerio Pascucci, et al. 2012. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *SC'12: Proceedings of the International Conference on High*

Performance Computing, Networking, Storage and Analysis. IEEE, 1–9.
[7] David Bernstein. 2014. Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing* 1, 3 (2014), 81–84.
[8] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015).
[9] Jieyang Chen, Qiang Guan, Zhao Zhang, Xin Liang, Louis Vernon, Allen McPherson, Li-Ta Lo, Patricia Grubel, Tim Randles, Zizhong Chen, et al. 2018. BeeFlow: A Workflow Management System for In Situ Processing across HPC and Cloud Systems. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1029–1038.
[10] Hank Childs, Sean D. Ahern, James Ahrens, et al. 2020. A Terminology for in Situ Visualization and Analysis Systems. *The International Journal of High Performance Computing Applications* 34, 6 (Aug. 2020), 576–691.
[11] Jai Dayal, Drew Bratcher, Greg Eisenhauer, Karsten Schwan, Matthew Wolf, Xuechen Zhang, Hasan Abbasi, Scott Klasky, and Norbert Podhorszki. 2014. Flexpath: Type-based publish/subscribe system for large-scale science analytics. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 246–255.
[12] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.
[13] James W Deardorff. 1970. A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers. *Journal of Fluid Mechanics* 41, 2 (1970), 453–480.
[14] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira Da Silva, Miron Livny, et al. 2015. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems* 46 (2015), 17–35.
[15] Nicola Demo, Marco Tezzele, and Gianluigi Rozza. 2018. PyDMD: Python dynamic mode decomposition. *Journal of Open Source Software* 3, 22 (2018), 530.
[16] 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.
[17] Bin Dong, Kesheng Wu, Surendra Byna, Jialin Liu, Weijie Zhao, and Florin Rusu. 2017. ArrayUDF: User-defined scientific data analysis on arrays. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. 53–64.
[18] Matthieu Dreher and Tom Peterka. 2017. *Decaf: Decoupled dataflows for in situ high-performance workflows*. Technical Report. Argonne National Lab.(ANL), Argonne, IL (United States).
[19] Kun Feng, Hariharan Devarajan, Anthony Kougkas, and Xian-He Sun. 2019. NIOBE: An Intelligent I/O Bridging Engine for Complex and Distributed Workflows. In *IEEE International Conference on Big Data*.
[20] Apache Software Foundation. 2020. Apache Hadoop. https://hadoop.apache.org
[21] Apache Software Foundation. 2020. Apache Storm. https://storm.apache.org/
[22] Apache Software Foundation. 2020. Spark Streaming. https://spark.apache.org/streaming/
[23] Yuankun Fu, Feng Li, Fengguang Song, and Zizhong Chen. 2018. Performance analysis and optimization of in-situ integration of simulation with data analysis: zipping applications up. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. 192–205.
[24] William F Godoy, Norbert Podhorszki, Ruonan Wang, Chuck Atkins, Greg Eisenhauer, Junmin Gu, Philip Davis, Jong Choi, Kai Germaschewski, Kevin Huck, et al. 2020. ADIOS 2: The Adaptable Input Output System. A framework for high-performance data management. *SoftwareX* 12 (2020), 100561.
[25] Vivien Gueant. 2020. iPerf - The TCP, UDP and SCTP network bandwidth measurement tool. https://iperf.fr
[26] Amazon Inc. 2020. Amazon Kinesis - Process & Analyze Streaming Data - Amazon Web Services. https://aws.amazon.com/kinesis/
[27] Google Inc. 2020. Google Cloud Dataproc. https://cloud.google.com/dataproc
[28] Hrvoje Jasak, Aleksandar Jemcov, Zeljko Tukovic, et al. 2007. OpenFOAM: A C++ library for complex physics simulations. In *International workshop on coupled methods in numerical dynamics*, Vol. 1000. IUC Dubrovnik Croatia, 1–20.
[29] Anthony Kougkas, Hariharan Devarajan, Jay Lofstead, and Xian-He Sun. 2019. LABIOS: A Distributed Label-Based I/O System. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing - HPDC '19*. ACM Press, Phoenix, AZ, USA, 13–24.
[30] Jay Kreps, Neha Narkhede, Jun Rao, et al. 2011. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, Vol. 11. 1–7.
[31] J. Nathan Kutz, Steven L. Brunton, Bingni W. Brunton, and Joshua L. Proctor. 2016. *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*. SIAM, Philadelphia, PA.
[32] 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 (CLADE '08)*. ACM, Boston, MA, USA, 15.
[33] Xiaoyi Lu, Md. Wasi Ur Rahman, Nusrat Islam, Dipti Shankar, and Dhabaleswar K. Panda. 2014. Accelerating Spark with RDMA for Big Data Processing: Early Experiences. In *2014 IEEE 22nd Annual Symposium on High-Performance Interconnects*. IEEE, 9–16.
[34] Xiaoyi Lu, Dipti Shankar, Shashank Gugnani, and Dhabaleswar K Panda. 2016. High-performance design of apache spark with RDMA and its benefits on various workloads. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 253–262.
[35] 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.
[36] Parviz Moin and Krishnan Mahesh. 1998. Direct numerical simulation: a tool in turbulence research. *Annual review of fluid mechanics* 30, 1 (1998), 539–578.
[37] Raffaele Montella, Diana Di Luccio, and Sokol Kosta. 2018. Dagon*: Executing direct acyclic graphs as parallel jobs on anything. In *2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 64–73.
[38] OpenCFD. 2019. OpenCFD Release OpenFOAM v1906. https://www.openfoam.com/releases/openfoam-v1906/
[39] Steve Plimpton. 1995. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics* 117, 1 (1995), 1–19.
[40] Rahul Potharaju, Terry Kim, Wentao Wu, Vidip Acharya, Steve Suh, Andrew Fogarty, Apoorve Dave, Sinduja Ramanujam, Tomas Talius, Lev Novik, et al. 2020. Helios: hyperscale indexing for the cloud & edge. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3231–3244.
[41] Daniel A Reed and Jack Dongarra. 2015. Exascale computing and big data. *Commun. ACM* 58, 7 (2015), 56–68.
[42] Clarence W Rowley, Igor Mezić, Shervin Bagheri, Philipp Schlatter, and Dan S Henningson. 2009. Spectral analysis of nonlinear flows. *Journal of fluid mechanics* 641 (2009), 115–127.
[43] Peter J Schmid. 2010. Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics* 656 (2010), 5–28.
[44] Christopher Sewell, Katrin Heitmann, Hal Finkel, George Zagaris, Suzanne T Parete-Koon, Patricia K Fasel, Adrian Pope, Nicholas Frontiere, Li-ta Lo, Bronson Messer, et al. 2015. Large-scale compute-intensive analysis via a combined in-situ and co-scheduling workflow approach. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–11.
[45] Craig A. Stewart, Timothy M. Cockerill, Ian Foster, David Hancock, Nirav Merchant, Edwin Skidmore, Daniel Stanzione, James Taylor, Steven Tuecke, George Turner, Matthew Vaughn, and Niall I. Gaffney. 2015. Jetstream: A Self-Provisioned, Scalable Science and Engineering Cloud Environment. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure (XSEDE '15)*. Association for Computing Machinery, St. Louis, Missouri, 1–8.
[46] TACC. 2020. Stampede2 User Guide - TACC User Portal. https://portal.tacc.utexas.edu/user-guides/stampede2
[47] Kunihiko Taira, Steven L Brunton, Scott TM Dawson, Clarence W Rowley, Tim Colonius, Beverley J McKeon, Oliver T Schmidt, Stanislav Gordeyev, Vassilios Theofilis, and Lawrence S Ukeiley. 2017. Modal analysis of fluid flows: An overview. *Aiaa Journal* (2017), 4013–4041.
[48] 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.
[49] Orcun Yildiz, Jorge Ejarque, Henry Chan, Subramanian Sankaranarayanan, Rosa M Badia, and Tom Peterka. 2019. Heterogeneous hierarchical workflow composition. *Computing in Science & Engineering* 21, 4 (2019), 76–86.
[50] Andy B Yoo, Morris A Jette, and Mark Grondona. 2003. Slurm: Simple linux utility for resource management. In *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 44–60.
[51] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. 2016. Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM* 59, 11 (Oct. 2016), 56–65.
[52] Henrique C Zanúz, Bruno Raffin, Omar A Mures, and Emilio J Padrón. 2018. In-transit molecular dynamics analysis with Apache flink. In *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*. 25–32.
[53] ZeroMQ. 2020. ZeroMQ, an open-source universal messaging library. https://zeromq.org/get-started