INSTANT: A Runtime Framework to Orchestrate In-Situ Workflows

Feng Li$^{1[0000-0002-8505-5208]}$ and Fengguang Song$^{2,\star[0000-0001-7382-093X]}$

[1] Purdue University, Indianapolis, IN 46202, USA
`li2251@purdue.edu`
[2] Indiana University Purdue University, Indianapolis, IN 46202, USA
`fgsong@iupui.edu`

Abstract. In-situ workflow is a type of workflow where multiple components execute concurrently with data flowing continuously. The adoption of in-situ workflows not only accelerates mission-critical scientific discoveries but also enables responsive disaster predictions. Although there are recent studies on the performance and efficiency aspects of in-situ workflows, the support for portability and distributed computing environments is limited. We present INSTANT, a runtime framework to configure, plan, launch, and monitor in-situ workflows for distributed computing environments. INSTANT provides intuitive interfaces to compose abstract in-situ workflows, manages in-site and cross-site data transfers with ADIOS2, and supports resource planning using profiled performance data. We use two real-world workflows as use cases: a coupled wildfire spreading workflow and a computational fluid dynamics (CFD) workflow coupled with machine learning and visualization. Experiments with the two real-world use cases show that INSTANT effectively streamlines the orchestration of complex in-situ workflows, and its resource planning capability allows INSTANT to plan and carry out efficient in-situ workflow executions under various computing resource availability.

Keywords: in-situ workflow · scientific computing · high-performance computing · urgent computing.

1 Introduction

Workflows have been widely used to enable scientific discoveries in different domains. A workflow describes the sequence of operations and the data/control dependencies among the operations. Traditionally, data dependencies of workflows are facilitated with offline file transfers, however with the increasing amount of data in different scientific domains, there is a trend to pursue in-situ workflows, where multiple components execute concurrently, with data flowing continuously across the workflow's lifespan. Although some researchers may use "in-situ" to describe the situation where different components co-locate in the same computing environment to reduce data transfer overhead [1, 2], "in-situ" in this paper refers to "processing data as it is generated" as discussed in [3].

* Corresponding author.

There are continuous community efforts to support in-situ analysis for different application domains, one of which is the ADIOS2 project. ADIOS2 (the second generation of the Adaptable Input Output System [4]) provides applications with a generic interface to switch among multiple file-based or streaming-based data transport methods. Parallel applications can use ADIOS2 APIs to read or write multi-dimensional data, and their choices of underlying I/O engines (transport methods) can be delayed to the runtime, by providing an external XML configuration file. This adaptive design makes it easier to conduct in-situ analysis for traditional HPC applications. ADIOS2 allows a group of m MPI processes each writing to a portion of multi-dimensional domain space, and another group of n MPI processes reading concurrently with data layouts different from the writer processes. There are also an increasing number of domain applications that have recently adopted ADIOS2, such as OpenFOAM [5] (computational fluid dynamics) and LAMMPS [6] (molecular dynamics).

Although the ADIOS2 library itself provides a universal interface to pairwisely connect various applications such as simulation, analysis, and visualization, it lacks the ability to compose and manage complex in-situ workflows. The loosely-couple model of ADIOS2 allows domain scientists to focus on each individual component's performance and usability, however, there is no high-level control or view of a workflow as a whole. As a result, the performance of in-situ workflows cannot be properly captured, and the in-situ workflows have limited portability and reproducibility due to the hardcoded and low-level ADIOS2 configurations.

Cheetah is a software framework to create "campaigns" for coupled simulation-analysis-reduction (SAR) computations [7]. Although Cheetah utilizes ADIOS2 to couple multiple component applications, it focuses on searching for good runtime parameter combinations in a single site through parameter sweeping, and it lacks the ability to compose workflows with a general DAG-like layout. Traditional workflow systems use the high-level DAG (Directed Acyclic Graph) abstraction to describe a workflow and allow the components of a workflow to be executed orderly following the precedence specified in the DAG. However, unlike traditional workflow, in-situ workflows feature in-situ data dependencies, which require special handling from workflow systems [8]. The integration of in-situ workflow and traditional task-based workflow has recently been explored in PyCOMMPs and Pegasus workflow management systems [9, 10]. However, these two integrations both rely on the Decaf library [11] for in-situ data transports, such that the in-situ transfer is limited to a single HPC site.

In order to provide high-level composition and orchestration support for complex in-situ workflows in distributed computing environments, we design and implement a runtime framework called INSTANT. INSTANT takes in an abstract workflow that consists of ADIOS2-enabled components, and generates executable workflows for running on distributed computing resources. The resource planning capability of INSTANT allows an abstract in-situ workflow to be mapped efficiently on different platforms, or across multiple platforms, based on workflow characteristics gathered through performance monitoring. The execution engine of INSTANT then launches the components of the workflow to the mapped computing environments and configures dataflow correspondingly. The flexible configuration interface of INSTANT not only makes the in-situ workflows portable and easily reproducible, but also enables instant deployment of critical pipelines.

In our experiments, we use two high-impact real-world workflow applications as use cases: a wildfire spreading workflow, and a CFD workflow coupled with real-time machine learning and visualization. Experiment results show that INSTANT realizes flexible configurations of in-situ workflows and allows efficient executions of in-situ workflows under different resource availabilities.

To the best of our knowledge, this work makes the following contributions:

1. A runtime framework to compose, plan, launch, and monitor complex in-situ workflows across multiple distributed environments.
2. An customized *DataX* I/O engine that supports flexible data interactions for wide-area networks.
3. Use case studies and performance analysis of real-world in-situ workflows, including a wildfire spreading workflow and a real-time "CFD + machine learning/visualization" workflow.

In the rest of this paper, we introduce the general design of the INSTANT runtime system in Section 2. We show the experiments with two real-world use cases in Section 3 and discuss the related work in Section 4. We assess the limitations and practical design decisions in Section 5, and then conclude the paper in Section 6.

2 Methodology

INSTANT mainly includes two main components, a "mapper" and an "execution engine", as shown in colored boxes in the Figure 1. The mapper takes in an abstract workflow as input and decides how to map the components of the workflow to a diverse set of sites. Such decisions are then instantiated as the "executable workflow" in the figure. The executable workflow is launched by the "execution engine" to the selected computing resources, which can be a grid, a computer cluster, or the local execution environment. Besides orchestrating remote jobs, the execution engine also sets up dataflows between workflow components (either same-site or cross-site using ADIOS2), and collects performance data which are used in turn for resource planning.

The separation of resource planning and execution engine are also seen in traditional workflow systems. However components execute one-after-another in a traditional task-based workflow system, and the data dependency is typically realized as offline file transfers. In comparison, INSTANT targets in-situ workflows, where components execute concurrently and the data transfer is continuous data flow instead of one-time file transfers. INSTANT allows a workflow described similarly to traditional workflows as DAG, and it intelligently decides the placements of workflow components and sets up the ADIOS2-based dataflow.

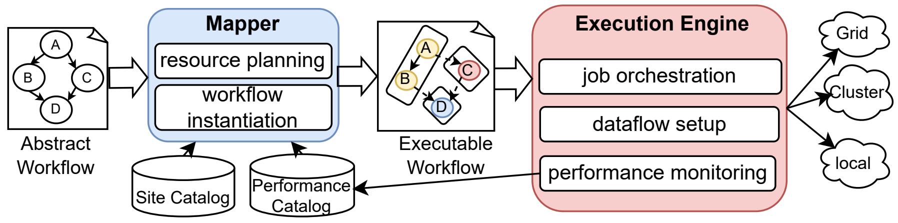


Fig. 1: Overview of the INSTANT runtime framework.

2.1 Mapper

The mapper takes in the abstraction of a workflow, site catalog and performance catalog as input information and generates an "executable workflow" as the intermediate result.

Workflow abstraction A workflow abstraction defines how each component is invoked for execution, and how data flows between components. The abstraction is resource independent, meaning that the same workflow can be executed on a local computer, a remote cluster, or a grid consisting of several clusters. The abstraction is designed in a way that a workflow user only needs to interact with the locally-install toolkit interfaces provided by INSTANT, without the need for preparing individual job scripts for remote submissions.

Listing 1 shows an abstract workflow description of a simple HeatTransfer workflow, which solves a 2D passion equation for temperature in homogeneous media using finite differences [12]. The HeatTransfer workflow contains two components, and the data writer runs iteratively and sends data to the reader continuously. In the abstraction file, the "name" field is the unique identifier of each component, and "exe" and the "args" fields describe the relative path of the component executable files and the runtime arguments, respectively. In the "dataflows" section, each entry describes a data flow between a pair of components. In this simple example, there is only a single data flow, which is from the "producer" component to the "consumer" component. The "IOFrom" and "IOTo" fields are the names of ADIOS2 IOs, and these IO names allow each component to initialize its IO engines based on the configuration of ADIOS2 XML configuration files provided later during runtime. As shown in Figure 2, the two IOs only allocate a "virtual" communication channel of two components. The corresponding engine choices for these IOs depend on the actual resource planning, which we introduce below.

Resource planning The resource planning utility decides where (which sites) and how (the number of processing units) to launch each component.

In an in-situ workflow, data continuously flows between components during the workflow lifetime in a pipeline fashion, and the overall speed of the workflow

```
{
    "components": [
        {
            "name": "producer",
            "exe":"heatTransfer_write_adios2",
            "args": ["adios2.xml", heat.bp, ...],
            "deployment": "$INSTANT_PATH/"
        },
        {
            "name": "consumer",
            "exe": "heatTransfer_read",
            "args": ["adios2.xml", heat.bp, ...],
            "deployment": "$INSTANT_PATH/"
        }],
    "dataflows": [
        {
            "componentFrom": "producer",
            "componentTo":"consumer",
            "IOFrom": "writer",
            "IOTo": "reader",
            "type": "Insitu"
        }]
}
```

Listing 1: The abstract workflow file for an example HeatTransfer workflow.

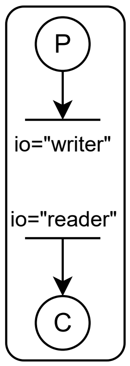


Fig. 2: The abstract workflow represented in Listing 1.

depends on the slowest segment [9,13]. INSTANT utilizes existing site catalog, collected performance data and resource planning to help an in-situ workflow to achieve better efficiency. The site catalog contains two parts:

– Compute-capability information: number of processing units (e.g. CPU cores) available at each site, and performance of each processing unit[3].
– Connectivity information: latency and bandwidth matrices between available sites.

The collected performance metrics mainly include the compute cost of each component and transfer sizes between components.

Currently, we utilize CPLEX as our default resource planning method. CPLEX together with its Optimization Programming Language (OPL) ([14,15]) allows us to define and solve the in-situ workflow optimization problem using a syntax similar to formal mathematical representations. The built-in optimization model optimizes the “throughput”, which is the number of steps the whole workflow can advance in a second. We first create the mathematical optimization model using OPL, respecting the actual resource limits and the pipelined execution constraints. Then CPLEX can build up a search space with reasonable combinations of different decision variables and search for the best solution. We have also developed a more efficient heuristic-based algorithm for the same optimization goal of maximizing workflow throughput, however we mainly discuss the CPLEX resource planning method in this paper for its simplicity.

[3] The per-processing-unit performance is currently recorded in the form of giga-floating-point operations per second (GFLOP/S).

The resulting resource plan is the decision on where to place each component, and how much computing resource to allocate for each component. Listing 2 then shows a possible launching plan of the previous HeatTransfer workflow. In the example workflow, the two components are assigned to two separate computing environments (PSC Bridges2 and IU Bigred 200) with different numbers of processing units respectively.

```
{
  "plans":[
    {
      "name": "producer",
      "site": "bridges2",
      "nprocs": 4
    },
    {
      "name": "consumer",
      "site": "bigred200",
      "nprocs": 2
    }
  ]
}
```

Listing 2: An example workflow plan file (cross-site plan).

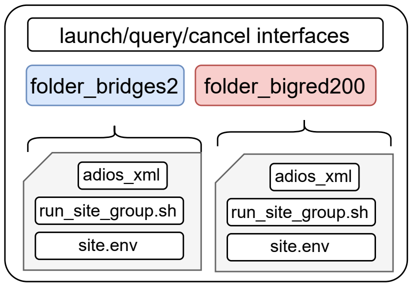


Fig. 3: Folder structure of an example executable workflow.

Instantiation Once the planning utility decides how to map each component, it can generate the "executable workflow". The executable workflow is an instantiation of the workflow plan and contains the required recipes to launch the workflow. The executable workflow is generated by first grouping components based on their site choices and then creating submission folders for each site group. Listing 3 shows the user interfaces of instantiating the executable workflow, where the "heat-transfer-dag.json" is the abstract workflow file, and "chosen-plan.json" is the plan file (either manually configured or generated by INSTANT resource planning utility). The output folder "testbed_folder" stores all generated contents of the executable workflow.

```
python3 scripts/instant_instantiate.py -c heat-transfer-dag.json -p chosen-
    plan.json -o testbed_folder
```

Listing 3: User interface to create a excutable executable workflow.

Figure 3 shows the contents of the output folder, where users can use the launch/query/cancel interfaces to orchestrate the remote executions of the insitu workflow. Two submission sub-folders are created for the two sites planned for workflow execution. Specifically, each site submission sub-folder includes a job script to invoke individual components assigned to the site (run_site_group.sh), an ADIOS2 configuration file to specify the choices of I/O engines (adios_xml), and an environment setup script (site.env).

The generated ADIOS2 XML configuration file allows the dataflow correctly configured in the later execution stage. Figure 4 shows how the gener-

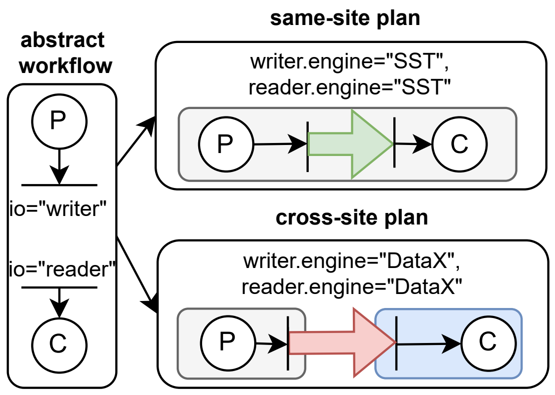


Fig. 4: INSTANT sets up either same-site or cross-site dataflow based on the plan.

ated ADIOS2 XML configuration file is used to prepare the workflows for same-site and cross-site launching. The original abstract workflow only defines the name of the ADIOS2 IOs, however, the actual transport method (the choice of ADIOS2 engine) is not determined until the planning is finished. In Figure 4, the "same-site" plan sets the engine type of both ends to "SST", which is the high-performance in-cluster transport provided by ADIOS2. In contrast, for a cross-site plan, the engine type is then configured as "DataX". The "DataX" is our customized ADIOS2 engine type to enable flexible data coupling across clusters, which we introduce later in this paper in Section 2.2.

2.2 Execution Engine

When the executable workflow is ready, the "execution engine" can launch the components to the target sites. The execution engine has three main goals:

– Job orchestration: launch, monitor and control the execution of remote jobs.
– Dataflow setup: set up and maintain the data communication channel for both in-site and cross-site dataflows.
– Performance monitoring: collect performance data which can then be used for resource planning to further improve the workflow execution efficiency.

Job orchestration As previously shown in Figure 3, the executable workflow exposes interfaces to launch, cancel and query the status of the workflow. The "launch" interface first copies the site-specific submission folders to the target sites, and then submits the site-specific job scripts (run_site_group.sh) to the HPC batch system. The query and cancel interfaces work similarly by issuing corresponding batch system job control commands. The submission folder copied to the target sites also contains the ADIOS2 XML configuration file, which allows components to set up different transports for its dataflow.

Dataflow setup At the start of remote execution, each component sets up its dataflow by initializing its I/O engines based on the specification defined in the ADIOS2 XML configuration file. Inside the ADIOS2 configuration file, each IO has the engine type specified, and the Adaptive I/O design of ADIOS2 allows the transports to be realized as either in-site or cross-site transfers.

The ADIOS2 library provides a universal view of scientific data and allows easy gluing of different applications using provided high-level language bindings. ADIOS2 provides several "engines" for different usage scenarios: the SST engine that max out transfer performance using high-performance interconnect; the DataMan engine that connects two endpoints across networks. We designed and developed a new engine called "DataX", which reuses the DataMan engine's ZeroMQ communication patterns, with the following features added:

1. Support arbitrary m-to-n process mapping.
2. Support scenarios that both sites are behind the firewall.

For feature #1, the current DataMan engine[4] only supports 1-to-1 process mapping (i.e., both DataMan producer and consumer have to use a single process), and DataMan is mainly used for cross-site communication between data transfer nodes of two clusters. In comparison, the more general ADIOS2 interface supports m-to-n process mapping: producer and consumer components can each be a group of MPI processes and have different access patterns of the global space. To provide such universal m-to-n process mapping for cross-site communication, our "DataX" engine adds additional support for data aggregation and redistribution for MPI ranks on both sides of the communication. This feature enables the support of the same flexible m-to-n process mapping even across wide-area networks, which allows a dataflow easily configured as same-site or cross-site.

For feature #2, the current implementation of DataMan requires the IP address and port of the reader to be accessible to the data writers, in order to establish the ZeroMQ data communication channel. However, it is common that HPC compute nodes are behind firewalls and not exposed to the outside of the institution, which makes it difficult to enable cross-site communication for in-situ workflows. For this reason, the INSTANT framework also includes a "relay" service, which creates endpoints in an accessible place that both ends can connect to. The relay service is implemented as an array of ZeroMQ "queue" devices[5] allocated in cloud virtual machines, which allows both sender and receiver to get connected even if they are both behind firewalls.

Performance monitoring Here we explain what information is needed for INSTANT to support resource planning, and how the required performance data is collected.

The resource planning utility mentioned in Section 2.1 requires several types of performance data to conduct resource planning: the per-step compute work of

[4] As of March 2023 when we submitted this work.
[5] ZeroMQ queue device: `http://api.zeromq.org/2-1:zmq-device`.

each component, the per-step transfer size of each data communication pair, and environment-related information such as latency/bandwidth between sites. We have added customized hooks for the *BeginStep* and *EndStep* ADIOS2 APIs, so that the ADIOS2 library automatically records the start and end time of each ADIOS2 step. For each component, the actual time spent on computing can be inferred from the elapsed time between the EndStep for reader operations and the BeginStep of the next write operations. The inferred compute time $T_{compute}$ and the documented performance data of the environments can then be used to calculate the actual computation work size $work = core\_speed \times num\_cores \times T_{compute}$. In the customized hooks, we also record the number of bytes written for each variable in each step.For component pairs that transfer multiple variables, the recorded transfer sizes are added together to obtain the per-step transfer size between the two components. The addition of customized hooks is transparent to workflow composers because the same set of ADIOS2 APIs are used. The bandwidth and latency information between HPC sites are obtained using iperftools and Linux ping command.

Our experiments in Section 3.2 demonstrate that through performance data collected from a previously-executed in-situ workflow, INSTANT can produce efficient resource plans for same-environment and cross-environment executions.

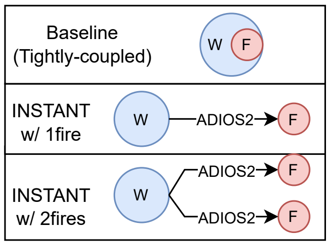
3 Use Cases

In this section, we show two real-world use cases of INSTANT. In the first use case we use WRF-SFIRE, a coupled atmosphere-fire model, and demonstrate how INSTANT can help accelerate the model coupling, and at the same time provide users with extensive flexibility/functionality. In the second use case, we use a real-time "CFD + machine learning/visualization" workflow, and show the advantage of INSTANT's resource planning and launching capability especially when computing resources are limited.

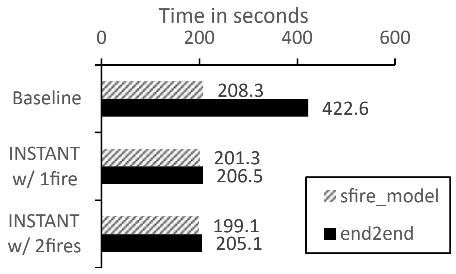
3.1 WRF-SFIRE

The first experiment uses WRF-SFIRE, which is a coupled atmosphere-fire model that is used for urgent simulations and forecasting of wildfire propagation [16]. WRF-SFIRE combines the state-of-the-art Weather Research and Forecasting model (WRF) and a surface fire spreading model (SFIRE). The atmosphere properties from WRF (e.g. surface air temperate, humidity, and wind) drive the SFIRE model, which then calculates the spreading of the fire line. The default/baseline WRF-SFIRE is a tightly-coupled model: the SFIRE model is implemented as one of the physics plugins of WRF, and WRF and SFIRE are built into the same binary executable. During runtime, the executable alternates between WRF and SFIRE models, which share the same memory space and CPU resources (i.e., time-division).

We compare the baseline tightly-coupled WRF-SFIRE method with the other two methods enabled by INSTANT, as shown in Figure 5a. Unlike the baseline method where WRF and SFIRE are tightly coupled, the INSTANT-enabled methods create decoupled in-situ workflows using ADIOS2. The "INSTANT w/ 1fire" method has two executable binaries: a WRF model without SFIRE component, and a standalone SFIRE executable. We developed the decoupled method based on the recent ADIOS2 IO backend for WRF [17][6]. Instead of utilizing the NetCDF for periodical variable output, the output data from WRF are sent out through ADIOS2 format for data streaming. For the data receiver side, we added ADIOS2 support for the standalone SFIRE by changing the default NetCDF I/O routines to corresponding ADIOS2 I/O routines. The "INSTANT w/ 2fires" method uses the same two executable binaries, but the WRF model sends data streams to two separate SFIRE simulations. This allows the workflow to use the same WRF output data to predict fire lines under different ignition conditions.



(a) Launching plans for WRF-SFIRE.



(b) Time comparison.

Fig. 5: Time comparison of the WRF-SFIRE workflow w/ and w/o INSTANT support.

For all three methods, we use the "hill" example included in the WRF-SFIRE repository, which simulates 5 minutes of the fire propagation in a 60m $\times$ 60m hill area. Figure 5b shows the time comparison with different execution methods. For each method, we plot the total end-to-end time (from the first step of the WRF model to the last step of the SFIRE model), and also the sfire_model time (elapsed time used for the SFIRE model execution). From Figure 5b, we can see that the default tightly-coupled method has the longest end-to-end time of 422.6 seconds. This lengthy time is caused by the time-division pattern of tightly-coupled execution: the same processors need to be time-sliced to alternate through WRF and SFIRE executions.

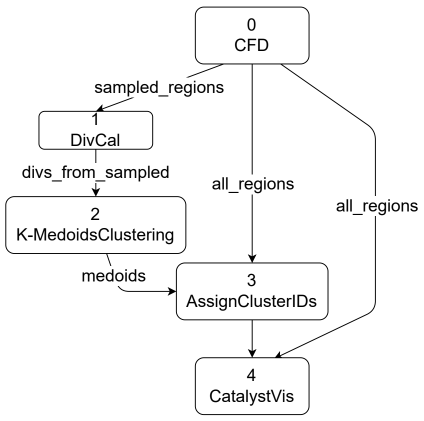
In comparison, the decoupled executions enabled by INSTANT deploy the WRF and SFIRE models in separate computing resources, and allow the data transfer to happen asynchronously without blocking the WRF atmosphere exe-

[6] The integration of ADIOS2 into the WRF is being added for future WRF releases `https://github.com/wrf-model/WRF/pull/1787`.

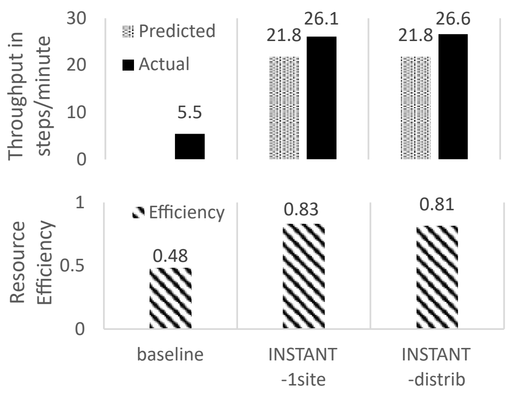
cution. For "INSTANT w/ 1fire" and "INSTANT w/ 2fires" methods, the total time is greatly reduced to 206.5 and 205.1 seconds, respectively, both resulting in more than 2 times speedup. In both cases, the end-to-end time is close to the time spent on the sfire_model. Moreover, compared with the base 1fire method, the 2fire method does more with similar time: two separate SFIRE models are concurrently executed, which gives more insights for disaster monitoring/prevention, without running the WRF atmosphere model multiple times. Overall, INSTANT enables flexible composition of in-situ workflows by allowing simulation connected with interchangeable analytics components.

3.2 Computational Fluid Dynamics with Real-time Machine Learning/Visualization

In the second use case, we use a real-time "CFD + machine learning and visualization" in-situ workflow application [18] to demonstrate how INSTANT can process execution patterns through collected performance data, generate efficient execution plans, and launch the proposed workflow to accelerate applications.



(a) Workflow layout.



(b) Throughput and resource efficiency.

Fig. 6: A CFD simulation + machine learning/visualization in-situ workflow. INSTANT achieves better throughput and resource efficiency through resource planning.

Figure 6a shows the workflow layout of the CFD-based workflow. The first *CFD* component application is a parallel icoFoam CFD simulation implemented with the OpenFOAM package to simulate a 2-D lid-driven cavity flow problem. Then the simulation output is partitioned into a number of 2D regions based on the geometric information, and the task is to cluster the regions into different categories based on the flow pattern. The *DivCal* component, to calculate the L2 divergences between a group of sampled regions. Then, the *K-MedoidsClustering* component groups all the sampled regions using k-medoid with the calculated divergence information. After that, the *AssignClusterIDs* component assigns a label for each region, based on its divergence from the medoid regions. Finally the *CatalystVis* component visualizes the clustering results using the ParaView Catalyst in-situ visualization toolkit [19].

We configure a grid size of 1024×1024 for the CFD simulation and a region size of 16×16, which results in a total number of 4096 regions. We compare the following three cases: *baseline*, *INSTANT-1site*, and *INSTANT-distrib*. The *baseline* case is a reference execution plan, which uses small-size allocation just to gather performance data for resource planning[7]. The two other methods use the collected performance data from the baseline execution and generate plans for two different resource availability scenarios. The *INSTANT-1site* method assumes there is a total of 32 cores available on a single HPC site (IU Quartz HPC); while the *INSTANT-distrib* method assumes there are 32 cores available in a distributed environment (two HPC systems: IU Quartz and Bridges2, each with 16 cores).

We use throughput and resource efficiency as the metrics to compare the above three methods. Throughput is measured in "steps per minute", which corresponds to the speed the workflow can advance in a pipelined fashion. The resource efficiency, on the other hand, is calculated by:

$$E_{\text{resource}} = \frac{\sum_{c_i \in C} n_{c_i} p_{c_i} T_{\text{compute}}(c_i)}{(\sum_{c_i \in C} n_{c_i} p_{c_i}) T_{\text{step}}}$$

Here C is the set of all components of the workflow, n_{c_i} is the number of processing units allocated to component c_i, and p_{c_i} denotes the performance of each assigned processing unit. The T_{step} is the workflow step time, which indicates the time required for the whole workflow to advance a step in the pipelined fashion. The $T_{\text{compute}}(c_i)$ is the time a component c_i uses for compute work instead of idling caused by pipeline stall. Overall, a higher resource efficiency indicates that components are assigned with the proper amount of computing resources, and the whole workflow experiences less idling.

Figure 6b shows the throughput and resource efficiency of the CFD workflow of the three methods. The upper part of the figure shows the throughput of the three methods, where the base case has a relatively low throughput of 5.5 steps per minute. With the performance gathered from the base case, the INSTANT creates plans for the *INSTANT-1site* and *INSTANT-distrib* methods. For those two methods, INSTANT can first give rough predictions of the throughput even before the execution, based on the results of resource planning. After the actual launching, INSTANT achieves 4.75 and 4.83 times better throughput for the 1-site and distributed setups, respectively, compared with the baseline. SNL-based methods also achieve better resource efficiency than the baseline, as shown in the bottom part of Figure 6b. Overall, INSTANT can help workflow users conveniently gather performance data from historical runs, and generate adequate resource plans for efficient executions at different resource availabilities.

[7] We have used 4,2,1,2,1 processes for the 5 components, respectively.

4 Related Work

Traditional workflow management systems such as Pegasus [20] and Kepler [21] provide interfaces for workflow users to compose, launch and collect results for task-based workflows. In these works, the data dependencies are carried out through file transfers, and each task can only start only after all its predecessors finish. Resource planning methods have also largely been developed following such assumptions of execution precedence. Our INSTANT framework, however, assumes a different pipelined execution pattern, and allows for special resource planning methods to target the emerging in-situ workflows.

The Cheetah/Savanna workflow environment [7] is a toolset to automate performance studies of coupled simulation-analysis-reduction HPC computations. Cheetah is used to compose a campaign of workflows, considering large configuration space such as process placement and I/O choices, and Savanna is a runtime engine that orchestrates individual workflows on target platforms. Although Cheetah/Savanna workflow environment supports different HPC platforms, it focuses on the fine-grained performance study on each individual platform, and the effect of collaboration of multiple platforms is largely unexplored.

BeeFlow [8] is a workflow management system that supports traditional workflows and also workflows with in-situ data dependencies. It utilizes event-synchronization primitives to enforce in-situ workflow logic. BeeFlow replies on Docker containers for application deployment, and the execution is constrained to one site for a single run. In comparison, INSTANT allows native parallel component applications, and the applications can be planned and deployed across multiple sites for efficient executions.

5 Discussion

5.1 Co-allocation of Computation Resources & Queue Time Waste

Currently, different site groups of a workflow are submitted subsequently to the planned sites, and we assume the components can start execution at around the same time. In the case when an HPC site experiences long job queue waiting time, the job submitted to the other sites will wait during the ADIOS2 environment initialization until the delayed job starts. For mission-critical applications, allocation reservation or increasing job queue priority can also better ensure that applications be launched and started around the same time.

5.2 Application Deployment in Distributed Computing Environments

One challenge in supporting the flexible execution of in-situ workflows on various platforms is the software deployment of component applications on different computing environments. To let components have the flexibility to be placed on either of the available sites, executables of the components should also be either available or deployable on those sites. Workflow systems such as Pegasus maintain a “transformation catalog”, to locate the executables for workflow components. Other systems use container technologies to deploy applications before execution. In our current implementation, we deploy component applications on the target sites using Spack environment [22]. INSTANT specifies a list of required software packages (e.g., OpenFOAM and ParaView) as a Spack environment file, which allows the same set of software environments to be easily installed/reproduced on various platforms.

6 Conclusion

We design and implement a runtime framework called INSTANT, which allows for easy configuration, planning, and deployment of in-situ workflows across multiple execution environments. The INSTANT framework contains a mapper component and an execution engine. The mapper can generate efficient execution plans based on available computing environments and workflow characteristics, and the execution engine allows the execution workflow to be deployed either on one site or across multiple sites. We conduct our experiments with a wildfire-spreading workflow and a real-time “CFD with machine learning and visualization” workflow. Experiment results show that INSTANT allows easier composition of in-situ workflows and built-in resource planning functionality improves the workflow throughput and resource efficiency. Future work includes supporting more applications from various domains. This will also allow for a more thorough performance study for a broad set of applications and workflows.

Acknowledgements This work is supported by the National Science Foundation Award #1835817. We want to thank Greg Eisenhauer who provided useful suggestions and clarifications on ADIOS2 cross-site in-situ data transport, and also Michael Laufer who offered generous help and guidance on the WRF-ADIOS2 integration.

References

1. Janine C Bennett, Hasan Abbasi, Peer-Timo Bremer, et al. 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*, pages 1–9. IEEE, 2012.
2. Christopher Sewell, Katrin Heitmann, Hal Finkel, et al. 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*, pages 1–11, 2015.
3. Hank Childs, Sean D. Ahern, James Ahrens, et al. A terminology for in situ visualization and analysis systems. *The International Journal of High Performance Computing Applications*, 34(6):576–691, August 2020.
4. William F. Godoy, Norbert Podhorszki, Ruonan Wang, et al. ADIOS 2: The Adaptable Input Output System. A framework for high-performance data management. *SoftwareX*, 12:100561, July 2020.

5. Hrvoje Jasak, Aleksandar Jemcov, and Z˘eljko Tukovic. OpenFOAM: A C++ Library for Complex Physics Simulations. In *International Workshop on Coupled Methods in Numerical Dynamics*, page 20, IUC, Dubrovnik, Croatia, 2007.
6. Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1–19, 1995.
7. Kshitij Mehta, Bryce Allen, Matthew Wolf, et al. A codesign framework for online data analysis and reduction. *Concurrency and Computation: Practice and Experience*, 34(14):e6519, 2021.
8. Jieyang Chen, Qiang Guan, Zhao Zhang, et al. 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)*, pages 1029–1038. IEEE, July 2018.
9. Tu Mai Anh Do, Loïc Pottier, Orcun Yildiz, Karan Vahi, Patrycja Krawczuk, Tom Peterka, and Ewa Deelman. Accelerating Scientific Workflows on HPC Platforms with In Situ Processing. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 1–10, May 2022.
10. Orcun Yildiz, Jorge Ejarque, Henry Chan, Subramanian Sankaranarayanan, Rosa M. Badia, and Tom Peterka. Heterogeneous Hierarchical Workflow Composition. *Computing in Science Engineering*, 21(4):76–86, July 2019.
11. M. Dreher and T. Peterka. Decaf: Decoupled dataflows for in situ high-performance workflows. (ANL/MCS-TM-371), July 2017.
12. ORNL. ADIOS2 HeatTransfer workflow. `https://github.com/ornladios/ADIOS2/blob/release_28/examples/heatTransfer/ReadMe.md`.
13. Y. Fu, F. Li, F. Song, and Z. Chen. 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*, HPDC'18, pages 192–205. ACM, June 2018.
14. IBM. IBM ILOG CPLEX Optimization Studio OPL Language User's Manual. Technical Report Version 12 Release 8, 2017.
15. Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250, 2018.
16. J. Mandel, J. D. Beezley, and A. K. Kochanski. Coupled atmosphere-wildland fire modeling with WRF 3.3 and SFIRE 2011. *Geoscientific Model Development*, 4(3):591–610, July 2011.
17. Michael Laufer and Erick Fredj. High performance parallel i/o and in-situ analysis in the wrf model with adios2. *arXiv preprint arXiv:2201.08228*, 2022.
18. Feng Li and Fengguang Song. Building a scientific workflow framework to enable real-time machine learning and visualization. *Concurrency and Computation: Practice and Experience*, 31(16):e4703, 2019.
19. Nathan Fabian, Kenneth Moreland, et al. The ParaView Coprocessing Library: A scalable, general purpose in situ visualization library. In *2011 IEEE Symposium on Large Data Analysis and Visualization*, pages 89–96, October 2011.
20. Ewa Deelman, Karan Vahi, et al. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, May 2015.
21. Bertram Ludäscher, Ilkay Altintas, Chad Berkley, et al. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice and Experience*, 18(11):1039–1065, 2006.
22. Todd Gamblin, Matthew LeGendre, Michael R Collette, et al. The spack package manager: bringing order to hpc software chaos. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2015.