In Situ Workflows at Exascale: System Software to the Rescue

Matthieu Dreher
Argonne National Laboratory
Lemont, IL, USA
mdreher@anl.gov

Swann Perarnau
Argonne National Laboratory
Lemont, IL, USA
swann@anl.gov

Tom Peterka
Argonne National Laboratory
Lemont, IL, USA
tpeterka@mcs.anl.gov

Kamil Iskra
Argonne National Laboratory
Lemont, IL, USA
iskra@mcs.anl.gov

Pete Beckman
Argonne National Laboratory
Lemont, IL, USA
beckman@mcs.anl.gov

ABSTRACT

Implementing an in situ workflow involves several challenges related to data placement, task scheduling, efficient communications, scalability, and reliability. Most of the current implementations provide reasonably performant solutions to these issues by focusing on high-performance communications and low-overhead execution models at the cost of reliability and flexibility.

One of the key design choices in such infrastructures is between providing a single-program, *integrated* environment or a multiple-program, *connected* environment, both solutions having their own strengths and weaknesses. While these approaches might be appropriate for current production systems, the expected characteristics of exascale machines will shift current priorities.

After a survey of the trade-offs and challenges of integrated and connected in situ workflow solutions available today, we discuss in this paper how exascale systems will impact those designs. In particular, we identify missing features of current system-level software required for the evolution of in situ workflows toward exascale and how system software innovations from the Argo Exascale Computing Project can help address those challenges.

CCS CONCEPTS

• **Computing methodologies** → **Concurrent programming languages**; • **Software and its engineering** → *Software reliability*; Data flow architectures;

KEYWORDS

Argo, MPI, In Situ Workflows, Exascale, System Software

1 INTRODUCTION

Many scientific processes can be expressed as a workflow graph where nodes are computational tasks (scientific simulations, analysis, visualization) and edges are data exchanges between tasks. Traditional workflow infrastructures exchange data through files. However, the increasing gap between I/O bandwidth and computational capabilities in current and future supercomputers requires a change in the way scientists are analyzing data produced by simulations. File-based workflows must now be replaced by in situ workflows performing data extraction, data reduction, or online visualization before storing relevant data to the file system.

Over the past few years, several in situ workflow solutions have been proposed by different research communities. Their designs tackle the same set of challenges related to data placement, task scheduling, efficient communications, scalability, and reliability. Nevertheless, these communities made different design trade-offs, based on their target workloads and production platforms.

We distinguish two types of currently available workflow systems, depending on how a system fits its execution model inside a typical HPC production machine. **Integrated** workflows map all their tasks inside a single MPI environment, providing a view of the workflow as a single program, and performing communications inside a single view of the entire execution. **Connected** workflows, instead, separate tasks into different executables, effectively partitioning a resource allocation among different pieces and communicating across distinct namespaces through explicit connections. Both types of in situ workflow designs have their strengths and weaknesses. Integrated workflows are easier to map inside a single MPI allocation on production systems, but connected workflows are more flexible to changes in the system and to faults in particular.

We argue here that, for exascale platforms, the design trade-offs of current in situ workflows systems need to be re-evaluated, most notably regarding reliability and flexibility, and that new features of system-level software can help with the resulting evolution.

This paper is organized as follows. We review current in situ infrastructures in Section 2, classifying them as either integrated or connected systems. We then discuss the typical challenges involved in the design of an in situ workflow system in Section 3. Section 4 details how those challenges are tackled by current implementations, and how exascale might impact those design choices. From these observations, we argue in Section 5 that system software can

help resolve the identified issues, and we focus in particular on new features of the Argo Exascale Computing Project.

2 RELATED WORK

Building an in situ infrastructure includes making several critical choices, starting at the execution model.

2.1 Integrated Workflow Infrastructures

Integrated workflow infrastructures include all their tasks within a single MPI program sharing a global communicator. We distinguish two major designs here. The first uses a host code, typically a simulation code, to run the analysis tasks within its MPI context. The analysis tasks take the form of a function call or a plugin to be loaded by the host code. That strategy was widely adopted by the visualization community. Current production visualization tools Paraview [3] and VisIt [2], and their respective in situ libraries Catalyst [16] and Libsim [22], insert rendering servers executed synchronously in time partitioning mode [1] within the simulation. The analysis tasks take the form of filters executed by the rendering server. ADIOS [17], a flexible I/O interface, enables data transformations [5] along the I/O path within the same MPI context as the caller. Damaris [10] adopts a space partitioning strategy. It splits the initial MPI communicator of the simulation in two groups, the simulation processes and the analysis processes, allowing the analysis task to run asynchronously from the simulation. The data transport method is selected separately from the tasks in an XML file. Some transport methods allow in situ computation to be performed in the same MPI program as the caller [5].

The second design, followed by Swift/T [23], organizes a pool of MPI processes as workers. Tasks taking the form of a function call are then loaded and executed by the runtime. Swift/T uses its own programming language to describe the workflow graph and extract parallelism. The runtime can then select precisely where to execute the tasks.

2.2 Connected Workflow Infrastructure

A connected workflow infrastructure coordinates tasks separated into different programs not sharing a common communicator. The in situ infrastructure must provide mechanisms to connect the tasks and create the appropriate communication channels. These frameworks rely on dedicated communication libraries to replace MPI, limiting portability but allowing more dynamic workflow graphs.

FlowVR [13] relies on a network of daemons to identify tasks and coordinate them. Each task requires minor modifications to connect to the local daemon and exchange messages with the rest of the application. DataSpaces [7] acts as a distributed data store. Applications connect to the server to publish or retrieve indexed data. The dynamic connections of tasks are managed by DART servers [8], which support high-speed interconnects such as InfiniBand. FlexPath [6], built on top of EVPath [14], embeds the connection information of all the tasks so that a new task can join the network and request the necessary connection information from any task.

Decaf [12] creates communication channels through MPI or CCI [4]. In the case of CCI, the addresses of the tasks are shared

through the file system during the startup phase of the workflow, enabling each task to connect to its dependent tasks.

3 IMPLEMENTATION CHALLENGES

When implementing an in situ infrastructure, integrated or connected, the developer must address several key challenges affecting the performance, portability, and usability of the designed infrastructure [11]. We highlight in this section some of these challenges and trade-offs and discuss how they can influence in situ infrastructure designs.

Addressing Tasks: To create communication channels between tasks, the in situ infrastructure must first be able to identify each task. This addressing mechanism can come from the underlying communication library (MPI) or from a service provided by the in situ infrastructure itself (naming servers, URIs).

Efficient Data Exchanges between Tasks: Each individual task may have its own communication needs. A workflow also creates additional communications to exchange data between tasks. These inter-task communications might generate interferences with the computation tasks, degrading their performance. Consequently, in situ infrastructures should provide efficient communication mechanisms in order to minimize their impact on the task performance.

User Code Integration: Scientific codes such as simulations and analyses can require the expertise of the developers to modify them. In situ infrastructures often require modifying the user code to integrate it within the infrastructure. These modifications should remain as limited as possible in order to ease the integration of user codes and promote adoption of the infrastructure.

Task Placement: Task placement is performance-critical for large-scale in situ workflows, particularly for data-intensive applications. Explicitly managing locality between tasks may deliver the best performance but is cumbersome and error prone and should be managed by the infrastructure automatically.

Resilience: In situ workflows may involve multiple tasks running on large scale supercomputers. The increase of computational resources involved in the workflow increases the probability of failures. Failures can come from multiple sources, for example, crash of a task, overflow in a communication channel, or failure of a node. Workflow engines should detect such faults without compromising the rest of the workflow and should apply corrective measures if possible.

Dynamicity: The graph of the workflow may have to change at runtime for different reasons: completion of a task, insertion of a new analysis, or temporary connection of a human to the workflow. Individual tasks might also require dynamicity in order to increase or reduce their computational resources, for instance, to reclaim the resources previously allocated to another task.

Portability: Supercomputers provide a wide range of specialized hardware, for example, GPUs, accelerators (Xeon Phi), and high-performance interconnects such as InfiniBand or Gemini. The developer must rely on specific libraries and runtimes to obtain the best performance on such hardware. However these libraries are often developed for a specific hardware and might not be portable to other platforms. Some libraries such as OpenCL or MPI provide a generic API supporting a broader spectrum of hardware but with lower performance than that of specialized libraries.

4 DESIGN CHOICES FOR CURRENT SOLUTIONS AND BEYOND

Current in situ infrastructures were developed to focus mainly on performance and scalability. We discuss in this section how these infrastructures address the other challenges presented in the preceding section. We also discuss how exascale system characteristics might shift the design priorities of future in situ infrastructures.

4.1 Integrated In Situ Infrastructures

Integrated in situ infrastructures host and execute the tasks of a workflow within a single MPI context. That model provides several advantages. First, MPI supports a large spectrum of high-performance interconnects and is available in almost all the super-computers on the Top500, providing very good portability. Second, MPI provides an easy mechanism to identify each task (MPI rank) and to create communication channels (communicators) between tasks. Third, the runtime has the flexibility to statically (Damaris, LibSim, Catalyst) or dynamically (Swift) place tasks at runtime and can therefore support different placement strategies. Fourth, the single MPI program model is the standard execution model for current supercomputer environments, making it easy to execute on today's platforms.

Yet the MPI model also has certain disadvantages. Because all the tasks share the same execution context, tasks are implemented as function calls. In some cases, the base code of one task, typically the simulation, is used to host the remaining tasks converted into function calls. In other cases, all the tasks must be converted into functions driven by one main program. That transformation might require significant code modifications and expertise in the user code. Additionally, MPI is not resilient to failures: a crash of a single task within the workflow causes a crash of the entire workflow.

4.2 Connected In Situ Infrastructures

Connected in situ infrastructures separate tasks in different executables. Since tasks no longer share a common MPI context, in situ infrastructures must replace some functionalities traditionally handled by MPI. In particular, infrastructures must provide a way to create communication channels between tasks and distribute computational resources to each task.

MPI_Connect would provide an answer to the first challenge, but its lack of support on current supercomputers necessitates other communication libraries, coming with their own challenges. First, communication libraries often provide a mechanism to address each communication point within the workflow, but it is up to the in situ infrastructure to share the connection information with the relevant tasks. Second, communication libraries such as DART [8], CCI [4], or Nessie [18] do not support or are not optimized for all high-performance interconnects, limiting the portability of the infrastructure.

In situ infrastructures must also deal with resource allocation. Because the tasks are not in the same execution context, it is more difficult to distribute the computational resources between the tasks unless the user provides more information to the infrastructure.

Despite these challenges, the connected mode enables several key features. First, it allows tasks to join and leave the workflow at runtime with their own computational resources. This implies that

the initial allocation of the workflow might grow or shrink at runtime and notifies the infrastructure of those changes. Unfortunately this feature is not supported by current production batch schedulers. Additionally, a connected in situ infrastructure can sustain the crash of a task without compromising the rest of the workflow. The user can then decide how to act on a task or node failure. For instance, EVPath calls a user-provided function to correct the workflow upon detection of the crash of a task. Moreover, the connected mode preserves the original user code and only requires minor modifications to enable the task to exchange messages with the rest of the application. This simplifies the integration process of complex codes into the infrastructure.

4.3 Exascale Is Coming

The need for performance, the simplicity of the MPI runtime, and the supercomputer environment constraints tend to favor the integrated mode for past and current solutions. Future exascale systems, however, will bring new constraints and challenges, causing developers to reconsider some aspects of current solutions.

First, the projected mean time to failure will decrease by a factor of 10 [9]. Individual tasks already provide a response to failures in some cases. However, this rate of failure will also compel in situ infrastructures to respond to these failures and adjust the workflow graph accordingly. Second, the number of cores per node will increase dramatically, and deeper memory hierarchies will make efficient placement of tasks more difficult. If tasks share a node, proper mechanisms for performance isolation will be required for both compute and memory. Third, we expect human-in-the-loop interactions to become more prevalent. Consequently the infrastructures will have to dynamically readjust the workflow graph and the distribution of its resources at runtime.

Connected workflows will be better tailored to face these new challenges. However, current supercomputer environments are not well suited for these infrastructures as they are more oriented toward integrated execution models. Yet these environments are also evolving to better support exascale systems and their workloads. For example, Argo, a system software Exascale Computing Project, is working on several features to bridge that gap.

5 HELP FROM THE SYSTEM-LEVEL SOFTWARE

The issues identified in the preceding sections indicate a lack of flexibility of the current HPC software stack, in particular from system software needed by in situ workflow infrastructures.

From a system software perspective, this lack of flexibility is due to a lack of advanced resource management mechanisms that would enable users and runtimes to build the right management policies. We define resource management here as the handling of the following issues.

Dynamicity: If we expect job allocations to be able to shrink and expand depending on several factors, including current power budget, node failures, and workload changes, then user-level software should be notified of these changes and be able to act on them. This goes both ways: user-level software should also be able to communicate to the system software changes in its resource requirements.

Node-level management: If a workflow supports multiple tasks on the same node or requires additional node services, then users should be able to partition the node resources among the current processes while taking into account the topology and hardware features of the node. This partitioning should also provide performance isolation between tasks.

Job-level management: Regardless of their execution model, workflows will have to configure and manage multiple types of tasks inside the same job, with possibly different configurations and different types of processes on subgroups of nodes.

Argo [19] is aimed at providing such advanced resource management services and making them directly available to users and runtimes. We describe here two of its components that will help future in situ workflows systems deal with those resource management issues.

5.1 Containers for Node Resource Management

The Argo NodeOS [21] is a set of extensions on top of the Linux operating system to provide resource partitioning mechanisms at the node level to HPC applications. It is designed around the idea of *compute containers*: a partition of the available resources where users can execute arbitrary commands, providing performance isolation from the rest of the processes. The operating system processes can be isolated in their own container, reducing the noise on the system. Furthermore, users can describe the resource requirement of a container in a declarative manner (e.g., this container requires 4 cores and 1 GiB of memory), and a system service called *node resource manager* will find a good partition for it.

The node resource manager also provides a local API to applications that can register themselves with the resource manager to be notified of changes on the node. For example, if the power budget of the node changes for administrative reasons, a workflow manager could react to this information by decreasing or increasing its workload [15].

We recently showcased how these compute containers can be used for performance isolation with in situ workflows [21]. In this setup, a molecular dynamics simulation (Gromacs) is coupled with an in situ visualization component (isosurface extraction) using a connected in situ middleware (FlowVR). Data exchanges between modules are performed by using a shared-memory space managed by the FlowVR daemon hosted on each node. The daemon is heavily multithreaded, consisting of four internal threads plus a thread for each module running on the node; none of them are computationally intensive.

Correct placement of application processes on the node is critical to obtaining optimal performance. The five in situ analytics processes together require at most 20% of the CPU cycles of a single core, but they must be kept apart from the Gromacs processes, which are highly sensitive to perturbations. Using compute containers, these different components sharing node resources can be isolated from each other easily, with the same performance as with a tedious manual placement. Instead, we can just declare the resource requirements of each component and have the node resource manager deal with partitioning.

5.2 Control Bus for Job-Level Resource Management

The Argo GlobalOS [20] is a set of user-facing distributed services that can be deployed inside job allocations or as part of the production infrastructure to control and monitor the resources available. It is based on the concept of *enclaves*: groups of resources that behave and can be controlled as a single entity.

As one of its core services, the GlobalOS provides a component called the *control bus*. This control infrastructure provides an API to partition a job into a hierarchy of enclaves and enables each enclave to execute distinct programs or be configured differently. This allows a native support for connected workflows, with each task living in a separate enclave. The control bus also enables the in situ infrastructure to register callbacks on resource events. Those callbacks will trigger for example when resource allocations shrink or expend. This allows in situ infrastructure to insert a new task or remove a finished one with their associated resources without having to modify the resource allocation of the other running tasks. Additionally, the in situ infrastructure can use the control bus to communicate directly with the underlying batch scheduler, to trigger those resource changes itself, using its internal knowledge of the current and future resource requirements of the workflow. For instance, in a human-in-the-loop scheme, this allows users to spawn new tasks and let the workflow infrastructure request the necessary resources automatically.

This infrastructure can thus be used by in situ workflow managers to perform space-partitioning or to launch different MPI subjobs. Using the control bus, one can also implement a naming service to communicate connection information between different subjobs and connect them back together. This strategy can replace file-based address exchanges, like in the case of Decaf with CCI.

Callbacks can also be registered with the control bus to trigger on failure events. The in situ infrastructure can then act on those events and implement fault recovery schemes. For example, failed tasks can be restarted on other resources or simply removed from the workflow. Alternatively, the in situ infrastructure can ask for additional resources to replace the failed ones.

5.3 Portability of Argo

While some of the features of Argo require changes in the production infrastructure (e.g., a different Linux kernel), the features relevant to in situ workflows discussed here are all in userspace. As such, they are intended to be portable to as many production systems as possible and can be deployed by users themselves.

By providing these new system-level features, we hope to simplify the implementation of connected workflows and allow users more flexibility in managing their allocations on production machines.

6 ACKNOWLEDGMENT

of Energy Office of Science and the National Nuclear Security Administration.

REFERENCES

[1] 2016. The In Situ Terminology Project. (Feb 2016). https://ix.cs.uoregon.edu/hank/insituterminology/index.cgi?n=Phase1B. Phase1BProposedInSituCategorizations.

[2] Sean Ahern, Eric Brugger, Brad Whitlock, Jeremy S Meredith, Kathleen Biagas, Mark C Miller, and Hank Childs. 2013. VisIt: Experiences with Sustainable Software. *arXiv preprint arXiv:1309.1796* (2013).

[3] James Ahrens, Berk Geveci, and Charles Law. 2005. ParaView: An End-User Tool for Large-Data Visualization. *The Visualization Handbook* (2005), 717.

[4] Scott Atchley, David Dillow, Galen Shipman, Patrick Geoffray, Jeffrey M Squyres, George Bosilca, and Ronald Minnich. 2011. The common communication interface (CCI). In *2011 IEEE 19th Annual Symposium on High Performance Interconnects (HOTI)*. IEEE, 51–60.

[5] D.A. Boyuka, S. Lakshminarasimham, Xiaocheng Zou, Zhenhuan Gong, J. Jenkins, E.R. Schendel, N. Podhorszki, Qing Liu, S. Klasky, and N.F. Samatova. 2014. Transparent In Situ Data Transformations in ADIOS. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 256–266. https://doi.org/10.1109/CCGrid.2014.73

[6] J. Dayal, D. Bratcher, G. Eisenhauer, K. Schwan, M. Wolf, Xuechen Zhang, H. Abbasi, S. Klasky, and N. 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 (CCGrid)*. 246–255. https://doi.org/10.1109/CCGrid.2014.104

[7] Ciprian Docan, Manish Parashar, and Scott Klasky. 2010. DataSpaces: an Interaction and Coordination Framework for Coupled Simulation Workflows. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10)*. ACM, New York, NY, USA, 25–36. https://doi.org/10.1145/1851476.1851481

[8] Ciprian Docan, Manish Parashar, and Scott Klasky. 2010. Enabling High-Speed Asynchronous Data Extraction and Transfer using DART. *Concurrency and Computation: Practice and Experience* 22 (2010), 1181–1204.

[9] Jack Dongarra, Pete Beckman, et al. 2011. The International Exascale Software Project Roadmap. *Int. J. High Perform. Comput. Appl.* 25, 1 (Feb. 2011), 58.

[10] Matthieu Dorier, Gabriel Antoniu, Franck Cappello, Marc Snir, and Leigh Orf. 2012. Damaris: How to Efficiently Leverage Multicore Parallelism to Achieve Scalable, Jitter-free I/O. In *CLUSTER - IEEE International Conference on Cluster Computing*. IEEE.

[11] Matthieu Dorier, Matthieu Dreher, Tom Peterka, Justin M Wozniak, Gabriel Antoniu, and Bruno Raffin. 2015. Lessons Learned from Building in Situ Coupling Frameworks. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*. ACM, 19–24.

[12] M. Dreher and T. Peterka. 2017. *Decaf: Decoupled Dataflows for In Situ High-Performance Workflows*. Technical Report ANL/MCS-TM-371.

[13] Matthieu Dreher and Bruno Raffin. 2014. A Flexible Framework for Asynchronous In Situ and In Transit Analytics for Scientific Simulations. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. https://hal.inria.fr/hal-00941413

[14] Greg Eisenhauer, Matthew Wolf, Hasan Abbasi, and Karsten Schwan. [n. d.]. Event-based Systems: Opportunities and Challenges at Exascale. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems (DEBS '09)*.

[15] Dan Ellsworth, Tapasya Patki, Swann Perarnau, Sangmin Seo, Abdelhalim Amer, Judicael Zounmevo, Rinku Gupta, Kazutomo Yoshii, Henry Hoffman, Allen Malony, Martin Schulz, and Pete Beckman. 2016. Systemwide Power Management with Argo. In *High-Performance, Power-Aware Computing (HPPAC)*.

[16] N. Fabian, K. Moreland, D. Thompson, A.C. Bauer, P. Marion, B. Geveci, M. Rasquin, and K.E. Jansen. 2011. The ParaView Coprocessing Library: A Scalable, General Purpose In Situ Visualization Library. In *2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. 89–96. https://doi.org/10.1109/LDAV.2011.6092322

[17] Qing Liu, Jeremy Logan, Yuan Tian, Hasan Abbasi, Norbert Podhorszki, Jong Youl Choi, Scott Klasky, Roselyne Tchoua, Jay Lofstead, Ron Oldfield, Manish Parashar, Nagiza Samatova, Karsten Schwan, Arie Shoshani, Matthew Wolf, Kesheng Wu, and Weikuan Yu. 2014. Hello ADIOS: The Challenges and Lessons of Developing Leadership Class I/O Frameworks. *Concurrency and Computation: Practice and Experience* 26, 7 (2014), 1453–1473. https://doi.org/10.1002/cpe.3125

[18] R. A. Oldfield, P. Widener, A. B. Maccabe, L. Ward, and T. Kordenbrock. 2006. Efficient Data-Movement for Lightweight I/O. In *2006 IEEE International Conference on Cluster Computing*.

[19] Swann Perarnau, Rinku Gupta, and Pete Beckman. 2015. Argo: An Exascale Operating System and Runtime. In *The International Conference for High Performance Computing, Networking, Storage and Analysis, SC15*.

[20] Swann Perarnau, Rajeev Thakur, Kamil Iskra, Ken Raffenetti, Franck Cappello, Rinku Gupta, Pete Beckman, Marc Snir, Henry Hoffmann, Martin Schulz, and Barry Rountree. 2015. Distributed Monitoring and Management of Exascale Systems in the Argo Project. In *IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS), Short Paper*.

[21] S. Perarnau, J. A. Zounmevo, M. Dreher, B. C. V. Essen, R. Gioiosa, K. Iskra, M. B. Gokhale, K. Yoshii, and P. Beckman. 2017. Argo NodeOS: Toward Unified Resource Management for Exascale. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 153–162. https://doi.org/10.1109/IPDPS.2017.25

[22] Brad Whitlock, Jean M. Favre, and Jeremy S. Meredith. 2011. Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization (EGPGV '11)*. Eurographics Association, 101–109.

[23] Michael Wilde, Mihael Hategan, Justin M. Wozniak, Ben Clifford, Daniel S. Katz, and Ian Foster. 2011. Swift: A Language for Distributed Parallel Scripting. *Parallel Comput.* 37, 9 (2011). https://doi.org/10.1016/j.parco.2011.05.005