The Challenges of Elastic In Situ Analysis and Visualization

Matthieu Dorier, Orcun Yildiz, Tom Peterka, and Robert Ross
Argonne National Laboratory
9700 Cass Avenue
Lemont, Illinois 60439
{mdorier,oyildiz,tpeterka,rross}@mcs.anl.gov

ABSTRACT

In situ analysis and visualization have been proposed in high-performance computing (HPC) to enable executing analysis tasks while a simulation is running, bypassing the parallel file system and avoiding the need for storing massive amounts of data. One aspect of in situ analysis that has not been extensively researched to date, however, is elasticity. Current in situ analysis frameworks use a fixed amount of resources and can hardly be scaled up or down dynamically throughout the simulation's run time as a response to changes in the requirements.

In this paper, we present the challenges posed by elastic in situ analysis and visualization. We emphasize that elasticity can take various forms. We show the difficulties of supporting each form of elasticity with the state-of-the-art HPC technologies, and we suggest solutions to overcome these difficulties. The resulting four-way classification can be seen as a taxonomy for future elastic in situ analysis and visualization systems.

CCS CONCEPTS

• **Computing methodologies** → **Parallel algorithms**; **Self-organization**; **Distributed algorithms**;

KEYWORDS

HPC, In Situ Analysis and Visualization, Elasticity

ACM Reference format:
Matthieu Dorier, Orcun Yildiz, Tom Peterka, and Robert Ross. 2020. The Challenges of Elastic In Situ Analysis and Visualization. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 6 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Over the past decade, in situ analysis and visualization have gained traction in the high performance computing (HPC) community as a means to speed up simulation campaigns. In situ analysis enables coupling analysis code with a simulation in order to bypass the storage system and allow direct insight into the simulation's data. This technique allows access to more data than would otherwise be available for postprocessing, and it opens the possibility for live user feedback. Major visualization and analysis toolkits such as VisIt [21] and ParaView [2] provide such capabilities.

One aspect of in situ analysis that has been seldom researched so far is that of elasticity, namely, the capability to dynamically add and remove resources to an in situ analysis framework to accommodate changes in requirements. Yet the case for elasticity has existed for as long as these frameworks have been around. For example, when we first proposed dedicating cores to in situ visualization tasks with Damaris back in 2013 [12], one concern scientists had was that computation resources would be wasted for the majority of the simulation's run time, during which it produces nothing interesting to visualize. The answer to this concern was to blame the simulation itself for its lack of elasticity: if the static, MPI-based simulation was not able to trade MPI ranks back and forth, how could an in situ analysis framework ever be elastic?

Since then, technology changes have made this answer less relevant. In particular, elasticity can take very different forms depending on *which resources* we want to add and remove and *when*, *why*, and *how* such an operation can occur. The increasing use of massively multicore nodes poses the question of better sharing these cores between simulations and analysis codes. New programming models, in particular based on user-level threads [30, 35], make it possible to share time even on individual cores. And while the MPI standard still constrains applications to a static set of ranks, workflows are trying to mitigate this limitation by allowing dynamic connections between applications.

Recognizing the various forms elasticity can take, in this paper we present four orthogonal categorizations for in situ analysis frameworks and discuss the challenges that the community has to overcome to enable each form of elasticity. The result is a taxonomy that we hope will highlight which form remains to be implemented and how close existing frameworks are to some forms of elasticity.

2 TAXONOMY AND CHALLENGES

In this section we present four classification axes of elastic in situ analysis techniques. These axes lead to a taxonomy for present and future in situ analysis systems.

2.1 *What?* Resources

Elastic in situ analysis techniques can be categorized by considering *which physical resources* are being added or removed during a reconfiguration operation. The main resources we consider in this paper are *cores* and *nodes*, although other resources could be considered, such as accelerators (GPUs, Tensor cores), local storage resources (SSDs, NVRAM), and network components. To understand the challenges posed by the trading of these resources, we

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

look at the software concepts that allow such trades: *threads*, *processes*, *jobs*, and *services*.

2.1.1 Threads. The increasing number of cores per socket has made multithreading almost mandatory to maximize performance on a node. Many threading/tasking frameworks are available to users, including Pthread [25], OpenMP [7], Intel TBB [35], and Argobots [30]. Some of these frameworks provide user-level threads (also known as coroutines), as well as custom schedulers and thread pools. Thread-level elasticity enables sharing cores, or even time on a core, between the simulation and the analysis code. It consists of having the analysis routines efficiently share a threading framework with the simulation. This level applies when the in situ analysis framework is tightly coupled with the simulation and lives in the same processes. Thread-level elastic in situ analysis can be enabled for example by (1) spawning more analysis threads when needed, (2) reducing the number of analysis threads when the simulation requires more threads for itself, and (3) enabling priority-scheduling of analysis and simulation threads depending on which type of work is more critical at any given time.

When simulation and analysis codes live in separate processes on the same node, synchronizing their use of individual cores becomes more difficult. The underlying threading framework is generally not shared across processes, and some additional communication mechanisms should be put in place for one process to notify another of a need to use more cores.

Challenges. While visualization libraries such as VTKm [23] have multiple threading backends (OpenMP and Intel TBB as of writing this paper), not all HPC threading frameworks are supported, hence leaving framework interoperability as a challenge. If the simulation uses Argobots, for example, VTKm will not be able to make use of the Argobots thread pools, and conversely the simulation will not be able to claim resources from VTKm. A possible solution would come in the form of a threading/tasking standard at a level lower than that of OpenMP, providing all the control over user-level threads and thread scheduling currently offered by Argobots and Intel TBB. Simulation and analysis codes would use this standard in place of concrete threading libraries, while threading libraries would serve as a basis for implementing the standard.

To date, no threading framework enables interaction across the boundaries of processes to share cores in a manner as efficient as what can be done within a single process. For example, if both a simulation and a colocated analysis code use OpenMP, a call to `omp_set_num_threads()` in one process will not trigger a change in the other. Solutions to this challenge would come either from extensions to threading libraries (with the problem of interoperability seen above) or from the operating system. Ideally, a process should be able to register callbacks to be called when other processes reduce the number of cores they use or request more cores.

2.1.2 Processes. Processes are a convenient software construct when trading cores or nodes. Many in situ analysis frameworks rely on MPI for parallelism (often in addition to threading). A process-level elastic in situ analysis framework should be able to accommodate for the addition and removal of MPI ranks. This does not mean that MPI itself should be elastic. One can imagine a scenario in which both the simulation and analysis code trade MPI ranks back and forth depending on dynamically changing requirements. If the simulation is bound to a static number of MPI ranks, process-level elasticity requires elasticity from MPI.

Process-level elasticity becomes more interesting if we consider that in situ analysis frameworks could be developed without MPI. RPC libraries such as Mercury [31] and even communication libraries such as libfabric [16] could be used directly to implement elastic analysis programs.

Challenges. The main obstacle to process-level elasticity is the fact that most in situ analysis frameworks today are base on MPI[1] and currently no application that we know of has the ability to change its number of MPI ranks at run time (although such a scenario could emerge from the field of HPC workflows [36]). Since MPI is not itself elastic, a solution using `MPI_Comm_spawn` could be envisioned to spawn the desired number of MPI processes for the analysis task. However, this MPI functionality is often not implemented and remains limited [13]. If the MPI standard enables some elasticity in the future, existing in situ analysis frameworks based on MPI would need to be updated to take advantage of it.

Going away from MPI by using alternative communication frameworks, on the other hand, means that a large number of algorithms will have to be reimplemented. Yet these algorithms already have benefited from years of testing and performance tuning in the context of MPI. Alternatively, new additions to the MPI standard such as MPI sessions [18] could provide a way of making a non-MPI distributed program temporarily become an MPI program for the duration of an analysis operation. The use of PMIx [6] to build an ad hoc MPI context is another possible solution, arguably more complex to implement.

2.1.3 Jobs. Job-level elasticity consists of being able to place in situ analysis programs in a job that is separated from the one in which the simulation runs. This approach is particularly appealing for interactive in situ analysis, where a user connects to a simulation from time to time to visualize its state. It also opens the possibility of running in situ analysis jobs in a machine distinct from that which runs the simulation.

Challenges. This approach implies that multiple jobs should be able to communicate with one another. On a single platform it requires that the job manager be informed that some jobs are running simulations while other jobs are running in situ analysis tasks for these simulations. The latter should have priority over the former (we want to enable in situ access to a running simulation rather than running multiple simulations without any possibility for in situ accesses) and ideally be scheduled as close as possible to the simulation on the network topology.

More complex challenges arise when multiple platforms are involved in the in situ analysis process, either at the same facility or across geographically distributed centers. In this context, these platforms are separated by different types of networks, involving different protocols. For example, the simulation may run on a supercomputer using a Cray Aries fabric, communicate through InfiniBand to gateways connecting it to a second supercomputer

[1] VTK actually provides abstract classes such as `vtkCommunicator` to implement parallelism. As of today only an MPI-based implementation is provided, but this abstraction is a good start for an implementation under another communication backend. The same applies to Damaris, which hides MPI-based communications under some `Reactor` and `Channel` abstract classes.

that relies on an Intel OmniPath fabric. Packets should therefore be routed and converted correctly across these different networks and their respective protocols.

With the convergence of HPC and cloud computing, the platform on which the in situ operations run may be a commercial cloud platform. Elasticity is a requirement in this scenario since the job scheduler on a given platform would hardly introduce dependencies on the availability of another platform that potentially belongs to another institution.

2.1.4 Services. Data services is an emerging approach that consists of replacing the file system with a service exposing the right interface and features for a specific family of applications [11, 38]. In the context of data services, in situ analysis consists of running analysis tasks inside the data service or *as a service*.

Challenges. HPC data services are often non-MPI programs. Hence, running legacy MPI-based analysis codes inside of them requires either creating a temporary MPI context using features proposed for future MPI standards, like MPI sessions, or rewriting analysis algorithms without using MPI. Given that these services are already tailored to specific applications, the latter is more likely, since the set of algorithms that need to be supported would be relatively small. The second challenge of service-level elastic in situ analysis is that of resource sharing with the service. Service-level in situ analysis can benefit from thread-level and process-level elasticity within the service, with all the challenges that these levels bring.

2.2 *When?* Time granularity

Elastic in situ analysis techniques can also be categorized by considering *when* a rescaling operation is allowed to happen. This leads to a notion of time granularity, for which we identify four categories. We named these categories *by the smallest entity that stays up and running* during the rescaling.

2.2.1 Platform. When the simulation and analysis codes run in distinct jobs, the analysis job may need to be terminated, and another job with a different size should be submitted in order to change the amount of resources that an analysis code uses.

Challenges. This granularity does not bring much of a challenge. The in situ analysis framework should guarantee that it can be safely shut down without impacting the simulation and can reconnect later, from another job. On the simulation side, the deployment of an in situ analysis job with a different size may mean that the output data are split in a different way from earlier instances of analysis jobs. Libraries such as Decaf [15], Bredela [14], and DIY [28] can help with such data redistribution problems.

2.2.2 Job. Making the distinction between a job (i.e., the set of resources allocated for a specific amount of time) and the application that runs inside a job, we can envision platforms that allow adding or removing nodes to/from a job. These jobs are sometimes called "malleable jobs" [17]. If the application inside a malleable job is not elastic itself, it should be terminated and restarted by using a different amount of resources.

Challenges. In this scenario, the challenge of connecting separate jobs is alleviated, but the challenges of ensuring safe reconnection and redistribution of data after restarting the in situ application remain. Additionally for batch jobs, one could envision a mechanism by which users indicate how an application should be terminated and restarted when the job changes size.

2.2.3 Application. The analysis application itself may be able to accommodate for a change in its resources; however, on-going analysis algorithms should first terminate (or abort) before making use of new resources or before the framework is able to release resources.

Challenges. If resources can be added to and remove from an application without shutting it down, this granularity mainly poses the challenge of data redistribution and synchronization with the simulation in the context of process-level elasticity. If the analysis application needs to change its number of threads or its use of cores by a threading framework, we can expect such a threading framework to make rescaling transparent to the simulation.

2.2.4 Algorithm. The finest granularity is that of an analysis algorithm. Namely, such an algorithm may be able to accommodate for changes in the amount of resources available without requiring a restart or cancellation. For example if the analysis algorithm processes a number of local blocks of data using a pool of threads bound to cores, the addition of a new thread can immediately be used to processes those blocks faster.

Challenges. We can expect multithreaded analysis algorithms to already be able to transparently accommodate a change in the number of threads or cores available. Challenges arise when considering processes and nodes. For example, with a sort-first parallel rendering algorithm, an attempt at reducing the number of processes would prompt leaving processes to send their partial results to remaining processes for them to carry on the rendering, rather than canceling the rendering task altogether to restart it on fewer processes. This type of granularity for process-level elasticity would make sense particularly for long-running analysis algorithms such as stream processing. We also note that some algorithms, for example based on the map-reduce paradigm, can enable this granularity without much change in the algorithm itself.

2.3 *Why?* Triggers

The third classification axis of elastic in situ frameworks focuses on *why* reconfiguration is triggered.

2.3.1 Manual triggers. Manual triggers consist of having a user request a reconfiguration. This is the most basic form of trigger and the easiest to implement. It suits particularly well scenarios where in situ analysis is done interactively and the need for reconfiguration comes from the need to run more or less expensive analysis tasks.

2.3.2 Simulation triggers. The simulation itself may request the analysis framework to scale up and down. This type of trigger does not present much of a challenge because it simply leaves to the simulation the task of choosing when and why reconfiguration is needed.

Challenges. Manual and simulation triggers do not pose much of a challenge, beyond providing the right API to enable such triggers.

2.3.3 Performance triggers. The system may try to reconfigure itself to maximize performance. For instance, the user may not

know how many threads are needed to perform a given analysis task in a minimum amount of time. An in situ framework accepting performance triggers is capable of monitoring its own performance and reconfiguring itself in consequence.

Challenges. The main challenge of performance triggers is the implementation of algorithms that can accurately model performance as a function of the resources allocated. These performance models can be developed offline or while the application is running.

2.3.4 Data triggers. The data being processed can itself be a trigger for reconfiguration. If the analysis framework is capable of determining the relative value of some data, it may automatically request more computation resources in order to run different analysis tasks. Data triggers may involve machine learning or deep learning to determine the value of data. They may also be combined with performance triggers so that the amount of allocated resources is determined to achieve the best performance given the data value.

Challenges. Machine learning algorithms will have to be developed to autonomously take decisions based on the value of the data.

2.3.5 External triggers. External triggers are requests from non-human entities external to the simulation/analysis compound. For example, an external trigger may come from the job scheduler, requesting the analysis framework to free up some resources in order to make space for another job. External triggers may be combined with performance triggers as well: the in situ analysis framework could interrogate the job management system to determine whether other jobs are competing for resources such as the network and to scale analysis tasks accordingly.

Challenges. External triggers pose the challenge of interfacing with external systems such as the job scheduler, either to collect information about other running jobs or to receive rescaling orders from these external systems.

2.4 *How?* Reconfiguration types

An additional axis can be added indicating the type of reconfiguration that is supported.

2.4.1 Upscaling and downscaling. Upscaling and downscaling respectively consist of adding and removing resources to an in situ analysis framework. One can imagine a system that can immediately use additional cores in the middle of an analysis operation but needs to wait until the end of an analysis operation for cores to be released.

Challenges. The challenges of upscaling and downscaling are related mainly to data redistribution. Solving this problem is easy when threads are the resources being added or removed. It becomes more complex when processes are involved. This challenge has already been described in preceding sections.

2.4.2 Topological reconfiguration. In the context of workflows more complex than a simple simulation/analysis pairing, topological reconfiguration consists of adding or removing analysis tasks, redistributing the resources between simulation and analysis, and changing the topology of the workflow and/or dataflow graph dynamically.

Challenges. Topological reconfiguration is arguably the most complex type of reconfiguration since it encompasses most of the challenges listed above: data redistribution, dynamic connection between distinct applications, and automatic response to triggers.

3 THE COMMUNITY SITUATION TODAY

In recent years, several in situ analysis solutions have been developed. However, support for elasticity remains largely unexplored in these systems. ADIOS [5], SENSEI [3], Decaf [15], ParaView Catalyst [2], LibSim VisIt [21], and Damaris [10] are all constrained by their dependency on MPI and their lack of dynamic multithreading support. Although some do use threads, they are not able to trade them with the simulation. TINS [9] makes one step forward in this respect. It is a task-based in situ framework that dynamically dedicates cores on each node for running the analysis processes. It relies on Intel TBB; and although it proved more efficient than static helper-core approaches such as Damaris, it requires the application to be written by using Intel TBB as well.

Some research efforts consider bringing elasticity to in situ frameworks. Flexpath is a publish/subscribe system for coupling workflow tasks and can accommodate analysis task arrivals/departures. However, the experiments by Dayal et al. [8] demonstrate only arrival/departure of serial analysis tasks with an MPI size of 1 to the static 2- or 3-task linear workflow. Melissa [33] is a parallel client/server architecture for the analysis of ensembles where independent simulation groups can connect dynamically to the parallel server when they start. Elasticity is one of the design goals of Melissa, although it is understood as the capability to run more or fewer simulation instances, rather than scaling up and down the in situ framework itself. Henson [24] is a cooperative multitasking system for in situ processing that uses position-independent executables and coroutines as its main abstractions. In a recent work, Lohrmann et al. [22] extended Henson with support for iterative workflows by allowing users to launch multiple jobs depending on the decisions made by one of the workflow tasks. This feature enables some elasticity at the job level.

When we look at the more general landscape of workflows, we find more studies on elasticity resulting from their inherently dynamic nature. FireWorks [20] is a workflow management system (WMS) for running high-throughput materials science calculations at distributed environments. It allows modification of the workflow graph during run time based on execution results. YAWL [34] supports dynamic workflows through worklets. Swift [36] is an implicit task-parallel language for scientific computing that supports elasticity through the evaluation of functions. Similarly, PyCOMPSs [32] is a Python-based distributed workflow system that enables users to program task-based parallel workflows and therefore supports elasticity. Embedding a simulation and an analysis program in one of the WMSs presented above can enable some form of elasticity: individual applications need to be restarted within the workflow, but the workflow itself keeps running.

One way of bringing elasticity to statically scheduled workflows would be to combine them with the distributed-area workflows that support elasticity. Pegasus has been combined with cloud infrastructures such as ExoGENI [4] and ORCA [29] to build elastic

distributed-area workflows. Yildiz et al. [37] employed Decaf workflows as single tasks of a PyCOMPSs workflow, which extended Decaf in situ workflows with elasticity.

Since MPI is widely used in the in situ workflows, bringing elasticity to MPI would play a key role in enabling elastic in situ analysis and visualization. Although the MPI standard provides dynamic process management via `MPI_Comm_spawn`, this feature is often not implemented: as of writing this paper, IBM and Cray's implementations of MPI do not support `MPI_Comm_spawn`. Additionally, this feature is not supported by current production batch schedulers. As an alternative to this feature, we proposed `MPI_Comm_launch` [13], which enables an MPI application to run inside another MPI application. This function may enable elasticity by dynamically starting subapplications. In a similar quest, Hori et al. [19] proposed process-in-process (PiP) which maps multiple processes into a single virtual address space (VAS). PiP defines a root process that owns the VAS, and it can spawn multiple arbitrary tasks executing in the same VAS.

Although some important studies of elastic in situ analysis and visualization have been made, elasticity still remains a long-term goal that will require community input from various groups. We refer the reader to a community effort defining related priority research directions [1, 26, 27].

4 CONCLUSION

Being able to reconfigure an in situ analysis framework at run time will be of outmost importance to enable better resource utilization at exascale. This elasticity has been requested by the community from the day in situ frameworks were proposed. In this paper we have shown that such elasticity can take different forms depending on the answers to four questions: which resource is being added or removed, when such a rescaling is allowed to happen, why it happens, and how. These various forms of elasticity present distinct technical challenges that the community will have to face in order to enable elasticity in existing and future in situ analysis frameworks.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357.

REFERENCES

[1] 2019. ASCR Workshop on In Situ Data Management. https://science.osti.gov/-/media/ascr/pdf/programdocuments/docs/2019/ISDM_brochure.PDF. (2019).

[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*. ACM, 25–29.

[3] Utkarsh Ayachit, Brad Whitlock, Matthew Wolf, Burlen Loring, Berk Geveci, David Lonie, and E Bethel. 2016. The SENSEI generic in situ interface. In *Proceedings of the 2nd Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization*. IEEE Press, 40–44.

[4] Ilya Baldin, Jeff Chase, Yufeng Xin, Anirban Mandal, Paul Ruth, Claris Castillo, Victor Orlikowski, Chris Heermann, and Jonathan Mills. 2016. ExoGENI: A multi-domain infrastructure-as-a-service testbed. In *The GENI Book*. Springer, 279–315.

[5] David A Boyuka, Sriram Lakshminarasimham, Xiaocheng Zou, Zhenhuan Gong, John Jenkins, Eric R Schendel, Norbert Podhorszki, Qing Liu, Scott Klasky, and Nagiza F Samatova. 2014. Transparent in situ data transformations in adios. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 256–266.

[6] Ralph H Castain, Joshua Hursey, Aurelien Bouteiller, and David Solt. 2018. PMIX: Process management for exascale environments. *Parallel Comput.* 79 (2018), 9–29.

[7] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: An industry-standard API for shared-memory programming. *Computing in Science & Engineering* 1 (1998), 46–55.

[8] 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.

[9] Estelle Dirand, Laurent Colombet, and Bruno Raffin. 2018. TINS: A task-based dynamic helper core strategy for in situ analytics. In *Asian Conference on Supercomputing Frontiers*. Springer, 159–178.

[10] Matthieu Dorier, Gabriel Antoniu, Franck Cappello, Marc Snir, Robert Sisneros, Orcun Yildiz, Shadi Ibrahim, Tom Peterka, and Leigh Orf. 2016. Damaris: Addressing performance variability in data management for post-petascale simulations. *ACM Transactions on Parallel Computing (TOPC)* 3, 3 (2016), 15.

[11] Matthieu Dorier, Philip Carns, Kevin Harms, Robert Latham, Robert Ross, Shane Snyder, Justin Wozniak, Samuel Gutierrez, Bob Robey, Brad Settlemyer, Galen Shipman, Jerome Soumagne, James Kowalkowski, Marc Paterno, and Saba Sehrish. 2018. Methodology for the rapid development of scalable HPC data services. In *Proceedings of the PDSW-DISC 2018 workshop (SC18)*. https://sc18.supercomputing.org/proceedings/workshops/workshop_pages/ws_pdsw106.html

[12] Matthieu Dorier, Roberto R. Sisneros, Tom Peterka, Gabriel Antoniu, and Dave B. Semeraro. 2013. Damaris/Viz: a nonintrusive, adaptable and user-friendly in situ visualization framework. In *IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*. Atlanta, United States. https://hal.inria.fr/hal-00859603

[13] Matthieu Dorier, Justin M Wozniak, and Robert Ross. 2017. Supporting task-level fault-tolerance in HPC workflows by launching MPI jobs inside MPI jobs. In *Proceedings of the 12th Workshop on Workflows in Support of Large-Scale Science*. ACM, 5.

[14] Matthieu Dreher and Tom Peterka. 2016. Bredala: Semantic data redistribution for in situ applications. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 279–288.

[15] Matthieu Dreher and Tom Peterka. 2017. *Decaf: Decoupled dataflows for in situ high-performance workflows*. Technical Report. Argonne National Laboratory, Argonne, IL (United States).

[16] OpenFabrics Working Group et al. Libfabric. https://www.openfabrics.org/. (????).

[17] Abhishek Gupta, Bilge Acun, Osman Sarood, and Laxmikant V Kalé. 2014. Towards realizing the potential of malleable jobs. In *2014 21st International Conference on High Performance Computing (HiPC)*. IEEE, 1–10.

[18] Daniel Holmes, Kathryn Mohror, Ryan E. Grant, Anthony Skjellum, Martin Schulz, Wesley Bland, and Jeffrey M. Squyres. 2016. MPI sessions: leveraging runtime infrastructure to increase scalability of applications at exascale. In *Proceedings of the 23rd European MPI Users' Group Meeting (EuroMPI 2016)*. ACM, New York, NY, USA, 121–129. https://doi.org/10.1145/2966884.2966915

[19] Atsushi Hori, Min Si, Balazs Gerofi, Masamichi Takagi, Jai Dayal, Pavan Balaji, and Yutaka Ishikawa. 2018. Process-in-process: techniques for practical address-space sharing. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 131–143.

[20] Anubhav Jain, Shyue Ping Ong, Wei Chen, Bharat Medasani, Xiaohui Qu, Michael Kocher, Miriam Brafman, Guido Petretto, Gian-Marco Rignanese, Geoffroy Hautier, et al. 2015. FireWorks: A dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience* 27, 17 (2015), 5037–5059.

[21] T Kuhlen, R Pajarola, and K Zhou. 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)*.

[22] Erich Lohrmann, Zarija Lukić, Dmitriy Morozov, and Juliane Müller. 2017. Programmable in situ system for iterative workflows. In *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 122–131.

[23] K. Moreland, C. Sewell, W. Usher, L. Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K. Ma, H. Childs, M. Larsen, C. Chen, R. Maynard, and B. Geveci. 2016. VTK-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE Computer Graphics and Applications* 36, 3 (May 2016), 48–58. https://doi.org/10.1109/MCG.2016.48

[24] Dmitriy Morozov and Zarija Lukic. 2016. Master of puppets: Cooperative multitasking for in situ processing. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 285–288.

[25] Bradford Nichols, Dick Buttlar, Jacqueline Farrell, and Jackie Farrell. 1996. *Pthreads programming: A POSIX standard for better multiprocessing*. " O'Reilly Media, Inc.".

[26] M Parashar, PT Bremer, K Heitmann, M Larsen, J Patchett, T Peterka, M Srinivasan, N Röber, S Frey, and B Raffin. 2019. 4.2 Workflow Specification. *In Situ*

Visualization for Computational Science (2019), 13.

[27] J Patchett, H Childs, A Bauer, PT Bremer, T Carrard, M Dorier, K Heitmann Garth, K Moreland, T Peterka, D Pleiter, et al. 2019. 4.3 Workflow Execution. *In Situ Visualization for Computational Science* (2019), 16.

[28] Tom Peterka, Robert Ross, Attila Gyulassy, Valerio Pascucci, Wesley Kendall, Han-Wei Shen, Teng-Yok Lee, and Abon Chaudhuri. 2011. Scalable parallel building blocks for custom data analysis. In *2011 IEEE Symposium on Large Data Analysis and Visualization*. IEEE, 105–112.

[29] Paul Ruth, Anirban Mandal, Yufeng Xin, Ilia Baldine, Chris Heerman, and Jeff Chase. 2012. Dynamic network provisioning for data intensive applications in the cloud. In *2012 IEEE 8th International Conference on E-Science*. IEEE, 1–2.

[30] Sangmin Seo, Abdelhalim Amer, Pavan Balaji, Cyril Bordage, George Bosilca, Alex Brooks, Philip Carns, Adrián Castelló, Damien Genet, Thomas Herault, et al. 2017. Argobots: A lightweight low-level threading and tasking framework. *IEEE Transactions on Parallel and Distributed Systems* 29, 3 (2017), 512–526.

[31] Jerome Soumagne, Dries Kimpe, Judicael Zounmevo, Mohamad Chaarawi, Quincey Koziol, Ahmad Afsahi, and Robert Ross. 2013. Mercury: Enabling remote procedure call for high-performance computing. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 1–8.

[32] Enric Tejedor, Yolanda Becerra, Guillem Alomar, Anna Queralt, Rosa M Badia, Jordi Torres, Toni Cortes, and Jesús Labarta. 2017. PyCOMPSs: Parallel computational workflows in Python. *The International Journal of High Performance Computing Applications* 31, 1 (2017), 66–82.

[33] Théophile Terraz, Alejandro Ribes, Yvan Fournier, Bertrand Iooss, and Bruno Raffin. 2017. Melissa: Large scale in transit sensitivity analysis avoiding intermediate files. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 61.

[34] Wil MP Van Der Aalst and Arthur HM Ter Hofstede. 2005. YAWL: Yet another workflow language. *Information systems* 30, 4 (2005), 245–275.

[35] Thomas Willhalm and Nicolae Popovici. 2008. Putting intel® threading building blocks to work. In *Proceedings of the 1st international workshop on Multicore software engineering*. ACM, 3–4.

[36] Justin M Wozniak, Timothy G Armstrong, Michael Wilde, Daniel S Katz, Ewing Lusk, and Ian T Foster. 2013. Swift/t: Large-scale application composition via distributed-memory dataflow processing. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 95–102.

[37] Orcun Yildiz, Jorge Ejarque, Henry Chan, Subramanian Sankaranarayanan, Rosa M Badia, and Tom Peterka. 2019. Heterogeneous hierarchical workflow composition. *Computing in Science & Engineering* (2019).

[38] Qing Zheng, Charles D Cranor, Danhao Guo, Gregory R Ganger, George Amvrosiadis, Garth A Gibson, Bradley W Settlemyer, Gary Grider, and Fan Guo. 2018. Scaling embedded in-situ indexing with deltaFS. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 30–44.