Lessons Learned from Building In Situ Coupling Frameworks

Matthieu Dorier
Argonne National Laboratory
Lemont, IL, USA
mdorier@anl.gov

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

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

Justin M. Wozniak
Argonne National Laboratory
Lemont, IL, USA
wozniak@mcs.anl.gov

Gabriel Antoniu
Inria Rennes Bretagne Atlantique
Rennes, France
gabriel.antoniu@inria.fr

Bruno Raffin
Inria, Univ. Grenoble Alpes
Grenoble, France
bruno.raffin@inria.fr

ABSTRACT

Over the past few years, the increasing amounts of data produced by large-scale simulations have motivated a shift from traditional offline data analysis to in situ analysis and visualization. In situ processing began as the coupling of a parallel simulation with an analysis or visualization library, motivated primarily by avoiding the high cost of accessing storage. Going beyond this simple pairwise tight coupling, complex analysis workflows today are graphs with one or more data sources and several interconnected analysis components. In this paper, we review four tools that we have developed to address the challenges of coupling simulations with visualization packages or analysis workflows: Damaris, Decaf, FlowVR and Swift. This self-critical inquiry aims to shed light not only on their potential, but most importantly on the forthcoming software challenges that these and other in situ analysis and visualization frameworks will face in order to move toward exascale.

Categories and Subject Descriptors

D.1.3 [**Software**]: Programming Techniques—*Concurrent Programming*; D.2.12 [**Software**]: Software Engineering—*Interoperability*; I.3.8 [**Computing Methodologies**]: Computer Graphics—*Applications*

Keywords

Exascale, In Situ Visualization, Simulation, Coupling, Damaris, Decaf, FlowVR, Swift

1. INTRODUCTION

Continued increases in high-performance computing (HPC) power and storage capacity combined with limited growth in I/O bandwidth are creating a crisis in the post hoc processing, analysis, and visualization of the data products of scientific simulations. Traditional techniques that perform analysis on disk-resident data must be replaced with in situ techniques that perform feature extraction, data reduction, or visualization on memory-resident data before accessing storage. Several approaches to inline analysis have emerged from different research communities: I/O, parallel visualization, and workflow management to name a few. Despite the progress that these groups have made, numerous challenges remain.

In order for in situ workflow systems to become mainstream exascale tools, discussions among developers of these tools are necessary to better understand the open challenges. This is precisely the goal of this paper. Beyond highlighting the strengths and limitations of the tools that we have developed over the past few years, we share lessons learned from using them on large-scale platforms and from interacting with end users, in the hope of guiding the community towards addressing remaining challenges.

In the following sections, we first provide a brief overview of the background of in situ analysis. We then discuss four coupling frameworks that we developed for in situ analysis and visualization: Damaris, which emerged from the development of an I/O middleware; FlowVR, which originated as a visualization pipeline for immersive systems; Decaf, which instantiates a dataflow for each link between producers and consumers in a workflow graph; and Swift, a scripting language to compose workflows from independently developed codes. We then present a list of lessons learned organized as follows:

- Data models and semantics: expressing the main requirements of the users' data requirements.
- Workflow management: the interoperability, placement, and resilience of task launching and execution.
- Programming models: dynamically launching and connecting tasks.
- HPC platforms: provisioning resources and services to monitor dynamic task execution.

We conclude with a set of open questions beyond the challenges presented in this paper.

ISAV '15 Austin, Texas USA
ACM 978-1-4503-4003-8/15/11 ...$15.00.
DOI: http://dx.doi.org/10.1145/2828612.2828622 .

2. BACKGROUND AND RELATED WORK

Solutions to in situ visualization and analysis are usually described by *where* the process is performed and *how* resources are shared. This has led to the common distinction between *in situ*, *in transit*, and *post hoc* visualization, as well as between *tight* and *loose* coupling. Another way to describe a system is in terms of its primary purpose: some solutions began as extensions of I/O libraries, while others were born from parallel visualization software, still others from workflow management systems (WMS). Many of these systems have grown over time to encompass more than their original objective.

If one considers in situ to be a replacement to offline file-based data analysis, it is natural to intercept the data on its way to the storage system. This can be done with little or no change to the code of existing simulations simply by overwriting or augmenting the I/O API that these simulations already use. ADIOS [1] is one example, which after providing a flexible I/O interface and data format, was extended to enable code coupling [35] and staging-area analysis [38]. GLEAN [15] is another example of a simulation/visualization coupling tool initiated by making HDF5 and PnetCDF I/O library calls. The HDF5 DSM file driver was also leveraged to enable inline visualization through an in-memory view of HDF5 datasets [5].

Another trend has been to consider that an HPC simulation is nothing but a source of data, just as files are. Consequently, parallel visualization and analysis backends were extended to directly expose an API that was originally hidden in their file readers. Examples of such in situ interfaces include VisIt's Libsim library [24], which provides a C and a Fortran interface to expose data to VisIt's engine, and ParaView Catalyst [20] (previously named co-processing library [13]), which relies on VTK data structures in C++.

A third trend is building workflows from several independent components. Workflow Management Systems include Pegasus [6], a WMS initially developed for grids, Taverna [26], Kepler [2], and Galaxy [14] for bioinformatics applications. Despite the prevalence of such tools, many users also still rely on scripting languages such as Python or Bash to launch and connect tasks.

3. OVERVIEW OF OUR FRAMEWORKS

In the following section, we briefly present four in situ analysis frameworks, highlighting key insights and features of each.

3.1 Damaris

Damaris[1] has been developed by Inria since 2011. Its initial objectives were to use dedicated cores to offload data processing tasks including compression, aggregation, analysis, and I/O [7]. Damaris emphasizes ease of use through a high-level API, external description of data in an XML format, and pluggable processing tasks written in C++ and Python. It was later extended to support dedicated nodes [34] and to provide a direct connection to the VisIt visualization tool with less code modification than calling Libsim directly [8].

Damaris's data management service is tightly coupled to the simulation. At startup, Damaris splits `MPI_COMM_WORLD` to dedicate some MPI ranks to data processing tasks. The communication of data is done either through MPI or shared memory (when configured to use dedicated cores).

Damaris was evaluated with the CM1 atmospheric simulation and the Nek5000 computational fluid dynamics code. It was also used in the OLAM simulation [16] and with a number of other codes by external researchers [21].

3.2 Decaf

Decaf[2] enables a hybrid approach of tight and loose coupling to implement the data flow between producer and consumer pairs in a workflow graph. The data flow consists of four primitives: selection, aggregation, pipelining, and buffering. Decaf is a software library for automatically constructing data flows from these primitives; it is designed to be a generic solution that other workflow and coupling tools can use.

Decaf integrates three levels of abstraction and software. A high-level *data description layer* captures data properties. The *data flow layer* optimizes the performance of each link between producer and consumer by implementing one or more data flow patterns. The *transport layer* moves data. Cross-cutting the three layers are low-overhead resilience strategies combining detection and recovery techniques for fault tolerance.

Decaf is a coupling service rather than a workflow system; its implementation is a C++ library whose API is called by higher-level workflow systems. One such simple scripting-based workflow language was developed in Python. The workflow graph is defined in a Python script, along with attributes on the nodes and edges of the graph. These attributes define resources to be allocated to producers, consumers, and data flows between them, as well as the callback functions to be executed during the execution of the workflow. The Swift scripting language, described in Section 3.4, is also a target workflow language for Decaf in a near future.

3.3 FlowVR

FlowVR[3] is a middleware designed to connect parallel applications in a workflow graph. The nodes of the graph are individual parallel or sequential executables, and the edges are communication channels. The graph is described through Python scripting, easily allowing creation of complex parallel communication patterns and workflows.

Originally intended for large-scale virtual reality, over the past several years FlowVR was redesigned to primarily support in situ applications. The user can specify precisely the placement (hosts, cores) of tasks to find the most efficient strategy (time sharing, helper core, staging nodes, hybrid) for a particular application. The FlowVR runtime then takes care of transmitting asynchronously the messages between tasks through shared memory or MPI.

FlowVR was evaluated with Gromacs, a molecular dynamics simulation code. FlowVR enabled steering capabilities [9], scalable in situ analysis and visualization [11], and was at the core of a complete framework for biological analysis [10].

3.4 Swift/T

Swift[4] [25] is a programming language to support massively scalable compositional programming. Swift has implicitly parallel data flow semantics, in which all statements are eligible to run concurrently, limited only by the data

[1] `http://damaris.gforge.inria.fr`

[2] `https://bitbucket.org/tpeterka1/decaf`

[3] `http://flowvr.sourceforge.net/`

[4] `http://swift-lang.org`

flow. Swift emphasizes a hierarchical programming model, in which *leaf tasks* linked to external libraries, programs, and scripts in other interpreted languages [27, 29] execute concurrently, coordinated by logic expressed in Swift code. Swift is typically used to express scientific workflows [37], controlling execution of relatively large tasks (seconds to hours); however, its high performance (1.5 B tasks/s on 512K cores [3]) allows it to be used as a high-performance computing language as well.

Swift/T [30] is derived from Swift/K, a grid workflow language. Swift/K is still supported and is based on the Karajan runtime, part of the Java CoG Kit [23], supporting wide area data movement and abstractions over many execution systems, such as PBS, SGE, Condor, Cobalt, and others. Swift/T maintains the core Swift language but is implemented entirely differently: it translates the Swift script into an MPI program for execution on a single machine.

Swift/T programs run on the Turbine runtime [28], which implements a small number of data flow primitives that enable Swift semantics on a scalable system without bottlenecks. Turbine is based on the Asynchronous Dynamic Load Balancer (ADLB) [17], a scalable master-worker system based on MPI [19]. Swift/T scientific computing applications have motivated myriad optimizations to enhance its performance on large-scale machines [33], including big data features [12, 32].

4. LESSONS LEARNED

Our experience with the four frameworks running applications on large-scale platforms, our interactions with users, and the limitations we encountered from the traditional HPC environments led us to a number of lessons described in the following sections.

4.1 Data Models and Semantics

While the trend to couple applications behind I/O interfaces facilitates a transparent coupling with legacy simulation codes, it may not actually be the best approach for describing data semantics. The reason is that analysis applications require more information than that available through I/O interfaces. For example, I/O interfaces do not express the fact that a particular variable represents the coordinates of mesh vertices onto which other variables are mapped. In addition to enabling analysis tasks, the ability to express such dependencies is critical in order for coupling frameworks to optimize data placement, task placement, and data transfers.

In addition to defining data dependencies and high-level semantics of the global data model, coupling frameworks and workflow management systems must be able to split, merge, and redistribute pieces of data while retaining their semantics. This is important for two reasons. First, analysis codes may not run on the same number of processes as simulation codes. Second, complex communication and data sharing patterns between tasks in a workflow may require splitting datasets into smaller pieces simply because of the large amount of data involved.

In this respect, Damaris requires users to supply additional semantic information (e.g., how to build a mesh from a set of coordinate variables, which mesh is associated with which field, etc.) but does not yet allow splitting or merging blocks of data. Damaris must rely on the domain decomposition provided by the simulation, whether or not this decomposition is appropriate for analysis tasks. FlowVR is also limited in terms of semantics: its simple put/get interface allows abstracting the communication layer but does not include any high-level information about the data. Thus, communicating tasks have to be designed with an implicit lingua franca, complicating interoperability between independent tasks.

The problem of data distribution is central in Decaf. Our solution is to annotate an existing data model with semantic information about the local or global nature of each field and how it is to be redistributed. There are a small number of ways that each field can be split or merged (e.g., round robin, z-curve, etc.), and these options are expressed as a flag for each field. The number of items to be considered as one indivisible unit is also provided (e.g., so that particle x,y,z coordinates remain together).

Swift presents a single address space to the script programmer, with typical data types such as integer, float, string, and binary string (blob), as well as *mapped* types that point to external data such as files or URLs. Rich data structures allow complex workflows to operate on structs and/or arrays of these types. Subtypes and typedefs of these types can be created, allowing the programmer to write highly readable programs; for example, the `file` type could be subtyped to `jpg_file`, or data structures of experimental data could be composed of studies, subjects, and observations [36]. Notably, Swift protects external execution, such as external program command lines, with strongly typed function signatures. This is in contrast to typical external execution mechanisms that accept a flat string (or array of strings). Thus, Swift prevents typical script programming errors caused by passing the wrong arguments to a given command.

4.2 Workflow Management

Critical challenges remain in the composition of previously and independently developed tools into flexible, automated workflows. In this direction, there are four main challenges.

Data dependencies in workflows are a general and powerful abstraction that allows the system to manage the concurrency of executing tasks. Supporting static and dynamic data dependencies in the programming model allows to express fine grain synchronization along the task graph. Thus, managing computational ensembles and dynamic events is a natural fit for data-dependent languages.

Damaris only provides an epoch-based data-dependency: the data output by a simulation is considered consistent only when the simulation has called `damaris_end_iteration` from all its processes. The graph of FlowVR applications natively describes the data dependencies between tasks. The user can have a fine control on the datastreams with the help of specific components to manage the data rate between two tasks. Additionally, some datastreams can be set as optional. This can be very useful for user-action driven datastream for instance. Swift programs make progress by resolving data dependencies and defining new data dependencies [33], which enables automatic parallelism. New data dependencies may be established in dynamic ways by evaluating conditional constructs, loops, and function calls. Decaf solves a different set of problems related to data flow, one level lower in the software stack than workflow management, and therefore does not need a notion of data dependency.

Interoperability is a particularly difficult challenge for complex scientific applications that were generally not developed to communicate with other applications. In addition to

the data semantics issues mentioned previously, more fundamental system issues concerning interfacing disparate programming languages and programmer assumptions about the operating system and execution environment must be resolved.

While it is easy to integrate Damaris in existing simulation codes, analysis tasks on the other hand have to be rewritten as plugins in C++ or Python. A FlowVR application assembles several executables written in C,C++ or Python to form a global applications. Each task should implement three light calls to the FlowVR API to receive and send data to the rest of the application. As such, it is quite convenient to integrate new user codes into an existing FlowVR application as the code modifications necessary are often just a few lines per task. Swift achieves interoperability through the command line interface for external execution; Swift/T additionally supports calls to C/C++ libraries wrapped with SWIG [4], Fortran libraries via SWIG+FortWrap [18]. Swift/T also provides high-level interfaces to optionally integrated Python, R, or Julia interpreters [29], and native code accessible via those languages (e.g., Numpy [22]).

Data locality and task placement are performance critical for large scale in situ analytics tasks, often very data intensive. Explicitly managing locality, conventional in single-program multiple-data (SPMD) programming, limits the automated concurrency made possible by data-dependent languages; ignoring data locality, on the other hand, lowers performance and makes it difficult to manage state.

The placement of tasks in Damaris (time-partitioning, using dedicated cores or dedicated nodes) is configurable by the user, but Damaris will neither attempt to optimize it nor to select the best one. In FlowVR the user can specify the hosts and the core affinity for each task to preserve the data locality. FlowVR can support both time and space partitioning strategies. However, static placement policies may require a strong effort from the developer. Though mitigated by the python scripting interface, it can be cumbersome for large heterogeneous applications. Data locality information may be used in Swift/T programs [12], enabling a blend of data-intensive computing features (e.g., MapReduce and its generalizations) with more general programming constructs.

Resilience is another important aspect of workflow management. Increasing the number of resources and interconnected tasks increases the probability of failures. These failures may result from a crash in part of a workflow, from overflow in communication channels connecting tasks that evolve at different rates, or from data corruption anywhere in the system. Different components of the workflow have different levels of resilience and require different responses to hard and soft faults.

Damaris does not provide any support for resilience. Pervasive use of data dependency processing in Swift gives the runtime rich information about workflow progress, enabling resilience features such as retry and restart (in Swift/K), and implies that Swift/T is well-positioned to utilize emerging fault-tolerance features on HPC machines. We expect to extend decaf to support resilience at the workflow level.

4.3 Programming Models

Our experience with developing in situ coupling frameworks has highlighted the mismatch between our overriding goal of connecting multiple applications and the main programming model available to implement them: MPI. MPI's legacy is communication within fixed-size single applications. Even though dynamic process management features have existed in the MPI standard since the outset, these features are rarely used, and HPC operating systems do not support them very well. Unfortunately, such features are of utmost importance for workflow management systems and coupling frameworks. Functions such as `MPI_Comm_connect / accept` are difficult to use (for example, they lack non-blocking versions) and are often not implemented or supported by the operating system.

These limitations in MPI forced us to implement Damaris as a library that splits `MPI_COMM_WORLD` at startup: our initial design involved an independent set of servers deployed once and reusable across different runs of a simulation. Likewise, Decaf splits `MPI_COMM_WORLD` into producer, consumer, and data flow communicators for each link in a workflow graph. The amount of overlap between communicators and physical cores is selectable in Decaf, permitting a range of tight to loose coupling. In practice, however, HPC operating systems today only support applications running on completely disjoint compute nodes, limiting Decaf's versatility. Swift scripts can issue tasks that are MPI libraries. These are supported by dynamically creating MPI subcommunicators from `MPI_COMM_WORLD` via the MPI 3 function `MPI_Comm_create_group()` [31]. Data is moved over MPI to support the functional Swift programming model, from task outputs to task inputs. Swift too is constrained by the external operating system, and must work within the original `MPI_COMM_WORLD`.

FlowVR doesn't follow the MPI model. Each parallel task is a separate executable which can be a MPI program but not necessarily. Each executable can be hosted on separate cores or node but can also be overlapped. However, current HPC operating systems on BlueGene and Cray do not allow the execution of multiple processes on the same core. Therefore FlowVR is incompatible with such machine and requires a full Linux cluster.

A second limitation of MPI comes from its datatypes. MPI datatypes are meant to capture the memory layout rather than data semantics. Such datatypes do not convey enough information for in situ analysis workflows. For this reason, they were ruled out from all our frameworks. Damaris transfers bulk data through shared memory and relies on its XML configuration file to attach semantics to the data. Decaf implemented its own datatype system and serializes its datatypes to byte buffers prior to communication. FlowVR relies on a put/get interface that only takes a pointer to raw bytes. Swift supports its data types (see Section 4.1), which include a pointer to raw bytes. More advanced type features based on MPI, HDF, or C function signatures are being considered.

In summary, the lack of elasticity in MPI and the lack of dynamic process management constrain the design of in situ workflow systems. Solutions that attempt to go beyond MPI (such as FlowVR) require a full Linux operating system to be supported. Additionally the lack of semantics in MPI datatypes led all our software to bypass them and transfer raw bytes while relying on their own mechanisms to manage semantics.

4.4 HPC Platforms

The last aspect that affects the potential of in situ analysis and visualization is the HPC environment itself. While workflow systems provide scientists the ability to define a graph of tasks and data dependencies of the (nonlinear)

process of scientific discovery, HPC platforms force them to submit a single *job* of fixed size and duration. This is counterproductive in several respects. First, if a user deploys an entire workflow in a single job, she will need to overprovision the job both in duration and in resources to accommodate unexpected behaviors of the workflow. Alternatively, breaking down the workflow into several pieces run sequentially as independent jobs forces each piece to checkpoint data to files, thus reverting to offline analysis.

These are topics where we, developers of in situ frameworks, rely on other research in the HPC community. The lessons learned here are a list of problems and wish list of solutions rather than results of our own experiments.

In this respect, the platform environment ought to support *dependent jobs* and *co-scheduling* to enable jobs to communicate and leverage common resources such as burst buffers, nonvolatile memory, and accelerators. We hope that such features will result from extreme-scale operating systems and runtimes research projects.

The second requirement, which will become even more critical if dependent and communicating jobs are enabled, is a *data management service*. Such a service should retain the data semantics and enable dynamic data placement strategies in response to a workflow's requirements (task dependencies and data dependencies). We hope that such a data management service will emerge from research related to parallel file systems.

5. BEYOND PRESENT CHALLENGES

Other challenges will need to be addressed in the longer term. Provenance and validation are frequently cited as important capabilities that future HPC frameworks will need to support to enable correct and reproducible scientific results. Bridging HPC with the outside world is another challenge: on one hand it will be necessary to find ways to transfer large amounts of data (input, output, and intermediate results) across facilities or cloud environments. On the other hand, extending workflows to connect HPC platforms with external sensors (such as particle accelerators) will be an important step to validate experiments against simulations. Another challenge is enabling steering and human interactions in the context of many-task workflows, not only to modify the parameters of a running simulation, but to dynamically add, remove or reconfigure tasks in a running workflow.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC02-06CH11357. This work was done in the framework of a collaboration between the KerData joint Inria - ENS Rennes - Insa Rennes team and Argonne National Laboratory within the Joint Laboratory for Extreme-Scale Computing.

6. REFERENCES

[1] H. Abbasi, J. Lofstead, F. Zheng, K. Schwan, M. Wolf, and S. Klasky. Extending I/O Through High Performance Data Services. In *Proceedings of the IEEE International Conference on Cluster Computing and Workshops (CLUSTER '09)*, Sept. 2009.

[2] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: an Extensible System for Design and Execution of Scientific Workflows. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pages 423–424, June 2004.

[3] T. G. Armstrong, J. M. Wozniak, M. Wilde, and I. T. Foster. Compiler Techniques for Massively Scalable Implicit Task Parallelism. In *Proc. SC*, 2014.

[4] D. Beazley. Automated scientific software scripting with SWIG. *Future Generation Computer Systems*, 19(5):599–609, 2003.

[5] J. Biddiscombe, J. Soumagne, G. Oger, D. Guibert, and J.-G. Piccinali. Parallel Computational Steering and Analysis for HPC Applications using a ParaView Interface and the HDF5 DSM Virtual File Driver. In T. Kuhlen, R. Pajarola, and K. Zhou, editors, *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2011.

[6] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A Framework for Mapping Complex Scientific Workflows Onto Distributed Systems. *Sci. Program.*, 13(3):219–237, July 2005.

[7] M. Dorier, G. Antoniu, F. Cappello, M. Snir, and L. Orf. Damaris: How to Efficiently Leverage Multicore Parallelism to Achieve Scalable, Jitter-free I/O. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER '12)*, Beijing, China, Sept. 2012. IEEE.

[8] M. Dorier, R. Sisneros, Roberto, T. Peterka, G. Antoniu, and B. Semeraro, Dave. Damaris/Viz: a Nonintrusive, Adaptable and User-Friendly In Situ Visualization Framework. In *Proceedings of the IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV '13)*, Atlanta, Georgia, USA, Oct. 2013.

[9] M. Dreher, M. Piuzzi, T. Ahmed, C. Matthieu, M. Baaden, N. Férey, S. Limet, B. Raffin, and S. Robert. Interactive Molecular Dynamics: Scaling up to Large Systems. In *International Conference on Computational Science, ICCS 2013*, Barcelone, Spain, June 2013. Elsevier.

[10] M. Dreher, J. Prevoteau-Jonquet, M. Trellet, M. Piuzzi, M. Baaden, B. Raffin, N. Ferey, S. Robert, and S. Limet. ExaViz: a Flexible Framework to Analyse, Steer and Interact with Molecular Dynamics Simulations. *Faraday Discuss.*, 169:119–142, 2014.

[11] M. Dreher and B. Raffin. A Flexible Framework for Asynchronous In Situ and In Transit Analytics for Scientific Simulations. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 277–286, May 2014.

[12] F. R. Duro, J. G. Blas, F. Isaila, J. Carretero, J. M. Wozniak, and R. Ross. Exploiting Data Locality in Swift/T Workflows using Hercules. In *Proc. NESUS Workshop*, 2014.

[13] N. Fabian, K. Moreland, D. Thompson, A. Bauer, P. Marion, B. Geveci, M. Rasquin, and K. Jansen. The ParaView Coprocessing Library: A Scalable, General Purpose In Situ Visualization Library. In *Proceedings of the IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV '11)*, 2011.

[14] J. Goecks, A. Nekrutenko, J. Taylor, et al. Galaxy: a Comprehensive Approach for Supporting Accessible, Reproducible, and Transparent Computational Research in the Life Sciences. *Genome Biol*, 11(8):R86, 2010.

[15] M. Hereld, M. E. Papka, and V. Vishwanath. Toward Simulation-Time Data Analysis and I/O Acceleration on Leadership-Class Systems. In *Proceeding of the IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV '11)*, Providence, RI, 10/2011 2011.

[16] R. V. Kassick, F. Z. Boito, P. Navaux, and Y. Denneulin. Investigating I/O approaches to improve performance and scalability of the Ocean-Land-Atmosphere Model. Presentation at the Seventh Workshop of the Joint INRIA/UIUC Laboratory for Petascale Computing, 2012.

[17] E. L. Lusk, S. C. Pieper, and R. M. Butler. More Scalability, Less Pain: A Simple Programming Model and its Implementation for Extreme Computing. *SciDAC Review*, 17, 2010.

[18] J. McFarland. FortWrap web site. http://fortwrap.sourceforge.net.

[19] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, 1994.

[20] ParaView. Catalyst. `http://catalyst.paraview.org`.

[21] F. Shahzad, M. Wittmann, M. Kreutzer, T. Zeiser, G. Hager, and G. Wellein. A survey of checkpoint/restart techniques on distributed memory systems. *Parallel Processing Letters*, 23(04), 2013.

[22] S. Van Der Walt, S. C. Colbert, and G. Varoquaux. The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2), 2011.

[23] G. von Laszewski, I. Foster, J. Gawor, and P. Lane. A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, 13(8-9), 2001.

[24] B. Whitlock, J. M. Favre, and J. S. Meredith. Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In *Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization (EGPGV '10)*. Eurographics Association, 2011.

[25] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster. Swift: A Language for Distributed Parallel Scripting. *Parallel Computing*, 37(9), 2011.

[26] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble. The Taverna Workflow Suite: Designing and Executing Workflows of Web Services on the Desktop, Web or in the Cloud. *Nucleic Acids Research*, 41(W1):W557–W561, 2013.

[27] J. M. Wozniak, T. G. Armstrong, D. S. Katz, M. Wilde, and I. T. Foster. Toward Computational Experiment Management via Multi-Language Applications. In *DOE Workshop on Software Productivity for eXtreme scale Science (SWP4XS)*, 2014.

[28] J. M. Wozniak, T. G. Armstrong, K. Maheshwari, E. L. Lusk, D. S. Katz, M. Wilde, and I. T. Foster. Turbine: A Distributed-memory Dataflow Engine for High Performance Many-task Applications. *Fundamenta Informaticae*, 28(3), 2013.

[29] J. M. Wozniak, T. G. Armstrong, K. C. Maheshwari, D. S. Katz, M. Wilde, and I. T. Foster. Toward interlanguage parallel scripting for distributed-memory scientific computing. In *Proc. CLUSTER*, 2015.

[30] J. M. Wozniak, T. G. Armstrong, M. Wilde, D. S. Katz, E. Lusk, and I. T. Foster. Swift/T: Scalable Data Flow Programming for Distributed-memory Task-parallel Applications. In *Proc. CCGrid*, 2013.

[31] J. M. Wozniak, T. Peterka, T. G. Armstrong, J. Dinan, E. L. Lusk, M. Wilde, and I. T. Foster. Dataflow coordination of data-parallel tasks via MPI 3.0. In *Proc. Recent Advances in Message Passing Interface (EuroMPI)*, 2013.

[32] J. M. Wozniak, H. Sharma, T. G. Armstrong, M. Wilde, J. D. Almer, and I. Foster. Big Data Staging with MPI-IO for Interactive X-ray Science. In *Proc. Big Data Computing*, 2014.

[33] J. M. Wozniak, M. Wilde, and I. T. Foster. Language Features for Scalable Distributed-memory Dataflow Computing. In *Proc. Data-Flow Execution Models for Extreme-Scale Computing at PACT*, 2014.

[34] O. Yildiz, M. Dorier, S. Ibrahim, and G. Antoniu. A Performance and Energy Analysis of I/O Management Approaches for Exascale Systems. In *Proceedings of the Sixth International Workshop on Data Intensive Distributed Computing (DIDC '14)*, pages 35–40, New York, NY, USA, 2014. ACM.

[35] F. Zhang, M. Parashar, C. Docan, S. Klasky, N. Podhorszki, and H. Abbasi. Enabling In-situ Execution of Coupled Scientific Workflow on Multi-core Platform. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS '12)*. IEEE, 2012.

[36] Y. Zhao, J. Dobson, I. Foster, L. Moreau, and M. Wilde. A notation and system for expressing and executing cleanly typed workflows on messy scientific data. *SIGMOD Record*, 34(3), 2005.

[37] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, I. Raicu, T. Stef-Praun, and M. Wilde. Swift: Fast, Reliable, Loosely Coupled Parallel Computation. In *Proc. Workshop on Scientific Workflows*, 2007.

[38] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, Q. Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. PreDatA – Preparatory Data Analytics on Peta-Scale Machines. In *Proceedings of the IEEE International Symposium on Parallel Distributed Processing (IPDPS '10)*, 2010.