Faodel: Data Management for Next-Generation Application Workflows

Craig Ulmer, Shyamali Mukherjee, Gary Templet
Sandia National Laboratories
Livermore, California
cdulmer|smukher|gjtempl@sandia.gov

Scott Levy, Jay Lofstead, Patrick Widener
Sandia National Laboratories
Albuquerque, New Mexico
sllevy|gflofst|pwidene@sandia.gov

Todd Kordenbrock
DXC Technology
thkorde@sandia.gov

Margaret Lawson*
Sandia National Laboratories
Albuquerque, New Mexico
mlawso@sandia.gov

ABSTRACT

Composition of computational science applications, whether into *ad hoc* pipelines for analysis of simulation data or into well-defined and repeatable workflows, is becoming commonplace. In order to scale well as projected system and data sizes increase, developers will have to address a number of looming challenges. Increased contention for parallel filesystem bandwidth, accomodating *in situ* and *ex situ* processing, and the advent of decentralized programming models will all complicate application composition for next-generation systems. In this paper, we introduce a set of data services, Faodel, which provide scalable data management for workflows and composed applications. Faodel allows workflow components to directly and efficiently exchange data in semantically appropriate forms, rather than those dictated by the storage hierarchy or programming model in use. We describe the architecture of Faodel and present preliminary performance results demonstrating its potential for scalability in workflow scenarios.

KEYWORDS

Workflow, composition, data management, scalability

ACM Reference Format:
Craig Ulmer, Shyamali Mukherjee, Gary Templet, Scott Levy, Jay Lofstead, Patrick Widener, Todd Kordenbrock, and Margaret Lawson. 2018. Faodel: Data Management for Next-Generation Application Workflows. In *ScienceCloud'18: 9th Workshop on Scientific Cloud Computing, June 11, 2018, Tempe, AZ, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3217880.3217888

*Also with Dartmouth College.

ScienceCloud'18, June 11, 2018, Tempe, AZ, USA

ACM ISBN 978-1-4503-5863-7/18/06…$15.00
https://doi.org/10.1145/3217880.3217888

1 INTRODUCTION

In order to accelerate the extraction of information from data, workflows that couple computational science applications are proving increasingly useful. By allowing developers to reason about their computational problems in a modular and regular manner, such workflows trade initial complexity in their definition and construction for ease of modification and increased reuse once they are in place. In order to maintain workflow construction as a viable technique for computational science, that complexity must be managed as projected system and data sizes increase.

Several potential complications are clouding the horizon, however:

- Impedance mismatches between data generation rates and parallel filesystem bandwidth have been an issue in computational science for some time now. System architectures have addressed this with various refinements to the memory hierarchy, more recently involving the development of node-local "burst buffers". While these changes provide more options and flexibility for workflow designers, they also introduce added design considerations.
- Coupling simulation and analysis using *in situ* and *ex situ* techniques present a different set of choices for workflow designers. Depending on system loads, problem inputs, or changing availability of specialized compute engines, it may be advantageous to migrate analytics tasks in and out of a particular application. While this may not make a difference in the end state of a workflow, adding this capability presents significant data management issues.
- While the bulk-synchronous-program (BSP) model remains dominant, emerging decentralized decomposition and scheduling models are being explored for their potential to provide dramatically increased scalability. Assumptions underlying the design of workflows for a group of cooperating BSP applications will not necessarily hold for workflows designed around asynchronous many-task (AMT) execution. For example, using a parallel filesystem to exchange data between workflow components will likely be problematic as the number of discrete application tasks increases.

Our group has been exploring how changing the ways in which cooperating applications exchange data can provide leverage on

these kinds of concerns. This paper describes a set of services for data management and exchange, Faodel, for use in such applications. Our goal is to allow workflow components to exchange data in forms whose semantics are dictated by the problem being solved by the workflow rather than by considerations such as the storage hierarchy or programming model in use. This paper contributes a discussion of the design and architecture of Faodel, a description of its integration into a specific application, and preliminary performance results in execution scenarios representing future workflow component patterns.

2 FAODEL DESIGN & ARCHITECTURE

The design of Faodel is largely driven based on requirements derived from the operating environment of today's high-performance computing (HPC) platforms. As depicted in Figure 1, Faodel provides a means of connecting several different jobs that run concurrently on a platform. First, BSP or AMT parallel simulation jobs run and produce data objects that are either stored in internal resources or published to other resources. While coupling and workflow scenarios may use Faodel to pass data between components, this more complex use case has not been fully explored yet. Second, Faodel may use distributed memory and NVM to absorb bursts of data from the application or replay results from simulations to requesters. Third, *in situ* analysis & visualization (ISAV) tools use Faodel to retrieve and analyze data. Finally, coordination applications provide a means of helping the different jobs in the environment locate and connect with the resources of other jobs.

An examination of the application environment motivated three fundamental requirements for the design of Faodel:

- Faodel must provide basic primitives for users to reason about and decompose their datasets, but at the same time the API must be as agnostic as possible about how developers manage their data. Rather than force users to design algorithms around a data store's indexing and migration policies, it is better to provide mechanisms for users to express how the system should manage their data. By offering mechanisms to control data epoch visibility and a simple key/blob interface, Faodel offers an approach that can serve many kinds of clients.
- To aid scalability, separating application fates (i.e., the simulation from the analytics) also offers independent scalability through loose coupling. This requires using a communication layer that offers efficient data transfers between jobs while at the same time not breaking the communication libraries used within jobs (e.g., MPI). As such, Faodel cannot simply rely on sockets or splitting an MPI communicator and must instead use a low-level Remote DMA (RDMA) communication layer. This layer is an evolution of the long proven NNTI layer from the Nessie RPC library [9, 13].
- Faodel must provide a way of migrating data objects from memory to higher-capacity resources, such as burst buffers or the parallel filesystem (PFS). This requirement implies Faodel must transition in-memory objects to systems with vendor-proprietary or file-based APIs.

Faodel is made up of several software components. We describe the higher-level components most relevant to application developers in the remainder of this section.

2.1 Kelpie

Kelpie provides a *key/blob* abstraction to facilitate flexible data exchange between different executables (e.g., simulation application and applications for visualization and analysis). A *key* is a programmer-defined text string that allows the programmer to attach semantic significance to the associated data, a *blob*. Although a key attaches programmer-cognizable meaning (and possibly structure) to a blob, Kelpie is entirely ignorant of any meaning attached to a key or its associated blob. An example key might encode the application name, run number, iteration number, variable name, and some information about what part of that globally distributed array this blob represents. Separate processes can exchange data via Kelpie by exchanging key information. Key exchange can be explicit or implicit (i.e., keys can be constructed in a well-known way).

2.2 Opbox

OpBox is a library for implementing asynchronous communication between multiple entities in a distributed application. Our experiences with remote procedure call (RPC) libraries found that while RPCs provide a simple way to coordinate data transfers and invoke action at remote nodes, it is often difficult to coordinate more sophisticated data management services (eg, ones involving more than two nodes, time-out conditions, or race conditions). Rather than leave the task of coordinating transfers entirely to the next layer up, OpBox provides the user with primitives for expressing a protocol as a state machine that the communication layer can process in an asynchronous manner. A communication pattern between a collection of nodes in OpBox is an `Op`. Users define and instantiate various `Op` classes as desired. Each provides a handle to the `Op` (implemented in C++ via `future`/`promise`), and a method to instruct OpBox to start running its encapsulated state machine. As each node in the `Op` communication pattern receives the `Op`, it processes the state machine accordingly.

Opbox provides a collection of operations that are common to many applications including ping and counter operations. In addition, Opbox includes a Directory Manager Service that can be easily incorporated into an application. The Directory Manager stores node information in a hierarchical directory. A typical Faodel application would have at least one Directory Manager instance acting as a naming service to locate components of an application.

2.3 Lunasa

Lunasa provides user-level memory management services for network memory. For performance reasons, Faodel relies heavily on RDMA to transfer data throughout an HPC system. RDMA eliminates the need to copy user data objects to kernel buffers and allows data transfers to occur without CPU intervention. RDMA transfers require the user register the memory buffers that are the source or destination of the transfer with the underlying network transport. Additionally, the virtual memory space that contains the memory buffer must be locked (or pinned) by the kernel to prevent it from

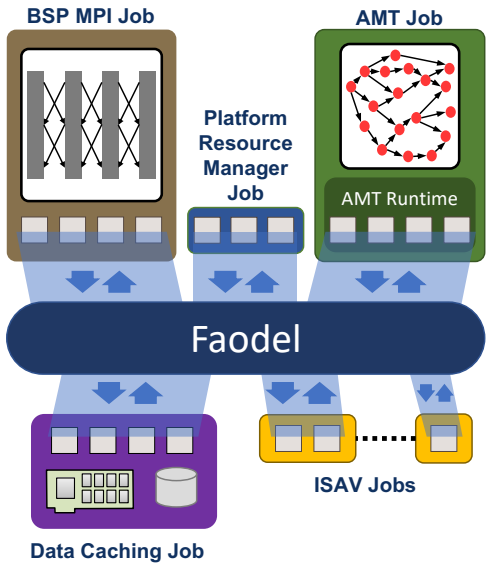

Figure 1: High-level diagram showing the relationship of Faodel to a workflow comprised of an AMT application and *in situ* analysis and visualization. The large colored shapes encompass the resources used by each entity. The small squares represent computational resources (e.g., compute nodes, processors, or threads). The cylinders represent storage resources.

being relocated in physical memory. The costs of registering and de-registering memory vary by network transport.

Lunasa amortizes these costs by facilitating the *explicit* reuse of registered memory. Existing approaches to reuse registered memory, *see e.g.*, [4, 8, 15], do so *implicitly*. In other words, instead of de-registering memory buffers at the end of an RDMA operation, the registered memory is left registered and its registration information is cached. Subsequent requests to re-register the cached memory can be fulfilled without incurring the costs of registration.[1] As a result, application programmers are encouraged to manage their memory buffers to extract as much reuse as possible. Lunasa frees application programmers from the burden of managing these implicit reuse semantics and allows them to *explicitly* request memory allocations for use in RDMA operations. To accomplish this objective Lunasa requests blocks of memory from the system and registers them at the outset. Lunasa then manages the resulting pool of registered memory as a standalone resource for satisfying user requests.

3 CASE STUDY: SIMPLEPIC

In this section we describe a sample application use case of Faodel services, enabling *in situ* analytics in a computational science code.

Particle in Cell (PIC) methods are a class of well-established computational techniques for simulating plasmas. Plasmas are comprised of charged particles in gaseous form interacting with each other and the surrounding environment. PIC methods simulate plasmas by computing: 1) *electromagnetic (EM) fields* created by charged particles in the plasma, 2) *chemical reactions* of plasma particles with the environment, and 3) *motion of particles* due to forces exerted by the EM fields. This last step generates currents and charges which further drive the EM field. Sandia is developing a new, PIC code named EMPIRE that is being architected to scale to next-generation computing platforms. While EMPIRE is currently implemented as a BSP-style MPI code, developers are evaluating whether better load balancing can be achieved through a DARMA-based AMT implementation. In order to allow researchers to better explore AMT tradeoffs, the EMPIRE developers have constructed a reference application called SimplePIC that isolates the particle *move* phase. This phase is especially appealing for an AMT environment because the move routinely causes imbalances in the way the data is distributed across compute nodes.

Whether in a BSP or AMT setting, PIC simulations generate a substantial amount of data that can make analysis challenging for two reasons. First, the volume of data managed by a simulation is large enough that writing to disk and performing post processing is infeasible. This characteristic drives the need for ISAV tools that can summarize current conditions in a way that is meaningful for users. Second, efficient PIC applications periodically redistribute their datasets to achieve better load balancing as particles disperse over time in the simulation. While this load balancing is performed manually in the BSP case and automatically in the AMT case, the end result is that downstream ISAV applications need a mechanism for locating and retrieving particles. Faodel performs this function in both cases. Examples of ISAV applications include tasks such as quantifying how many particles are close to regions of interest in the mesh, identifying how many particles exceed a threshold velocity, and rendering images to help developers verify the simulation is modeling an environment correctly.

In order to provide a portable way for connecting PIC and ISAV applications through Faodel, we have constructed a *Particle DIM*

[1] It is also worth noting that caching the registered memory eliminates the de-registration costs that would otherwise be incurred at the end of an RDMA operation.

(Data Interface Module) that is usable by both producers and consumers of PIC data. From the producer perspective, an application task periodically generates a *patch* of particle data that is injected into Faodel through the task's Particle DIM. The Particle DIM serializes the patch data into one or more key/blob pairs that are then published into Faodel's resources. The unique key names for patch data are also published as metadata in Faodel to provide a way for downstream applications to locate data. From the consumer's perspective, an application uses the Particle DIM to query metadata and retrieve relevant particle patches.

From the programmer's perspective, there are multiple advantages to using a DIM to interface with Faodel. First, a DIM establishes a contract between producers and consumers about how data is exchanged, but does not dictate how those transfers are implemented. This property allows DIM developers to write multiple implementations that decompose datasets in different ways if needed. Second, a DIM enables simulations to be decoupled from ISAV applications while retaining the ability to leave data objects in place if needed. The Particle DIM can be configured to store data objects in the application or to other distributed resources that are part of Faodel. Finally, a DIM provides a mechanism for implementing indexing that's right for the application. While the current Particle DIM performs basic indexing to locate items, it can easily be extended to allow consumers to make advanced queries. Our approach in this work is to have data producers compute statistics on data as it is inserted and store the information as additional metadata in Faodel.

4 EVALUATION

4.1 Opbox

One system that makes use of the Faodel infrastructure, and of Opbox in particular, is the custom metadata management system EMPRESS [7]. It allows clients to create custom attributes on parts of a variable, the whole variable, or even the whole data set to accelerate reading for data analysis. The two fundamental components of EMPRESS are a client API and a set of dedicated metadata servers. Opbox provides the connection between clients and servers. Each EMPRESS function has an underlying Op that passes the request and related metadata between the clients and servers. The performance of these Ops was tested using 100 write clients, 10 read clients, and one EMPRESS server. The data storage mechanism for the EMPRESS servers is currently a SQLite in-memory database. The evaluation results presented were obtained on the Chama capacity cluster at Sandia.

Testing indicates that each of the fundamental writing and reading Ops are efficiently supported as indicated in Figures 2 and 3. All times are from the client perspective, include the round trip communication and database operation times, and are the average of 5 experiments. In each experiment, each client performs one iteration of read or write operations. The operations demonstrated for writing are (0) "activating" an output set (committing a transaction), (1) the metadata associated with the entire run overall, (2) information for the current timestep, (3) the metadata for a single globally distributed variable, (4) creating a new, custom attribute type, (5) attaching an attribute to a run, (6) attaching an attribute to a timestep, and (7) attaching an attribute to part of a variable. For reading, the query returns (0) information about the run, (1) what timesteps are written for the run, (2) what variables are in a particular timestep, (3) what attributes have been attached to the run, (4) what attributes have been attached to the timestep, (5) what attributes have been attached to the variables, (6) attributes of a given type for a variable, and (7) attributes for a variable.

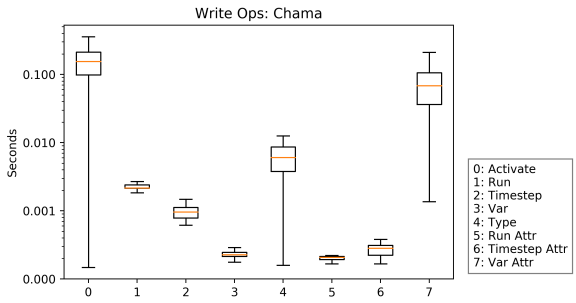

Figure 2: Writing operation total times

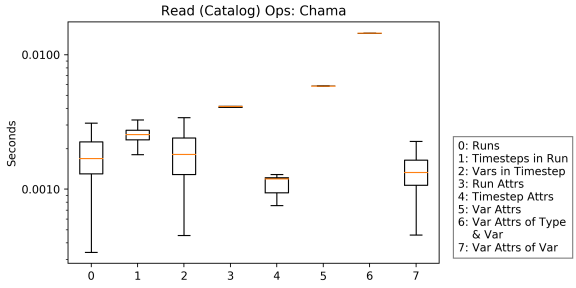

Figure 3: Reading operation total times

4.2 SimplePIC analytics

To demonstrate the feasibility of Faodel, we implemented a simple visualization example using SimplePIC and VTK [5]. In this example, particle data from SimplePIC are inserted into Faodel. The visualization application then retrieves data from Faodel and uses it to construct VTK objects for the purpose of generating a visual representation of the data.

The results of this experiment are shown in Figure 4. This preliminary evaluation is intended to demonstrate functionality and feasibility rather than performance. This figure shows the time required to move groups of particle data as a function of the number of particles being moved. The blue line shows the cost of inserting a block of particle data into Faodel. Because Faodel is designed to exchange data using shallow copies of reference-counted data structures, the time required to insert a block of particle data is independent of the number of particles for which data are being inserted (displayed as insignificant overhead in Figure 4).

The orange line in Figure 4 shows the cost of reading particle data from Faodel and generating VTK objects from its contents. In this case, the time required increases with the size of the data being manipulated. This is due to the fact that using particle data

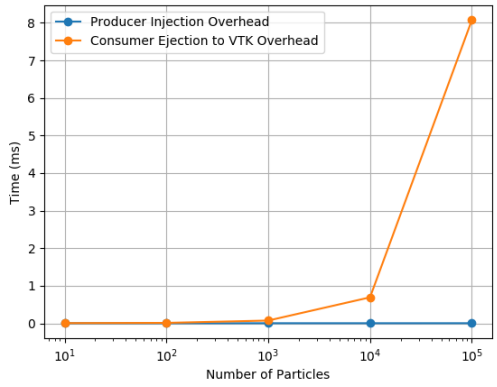

Figure 4: Preliminary data demonstrating the costs of moving data into and out of Faodel

to create VTK objects currently requires a deep copy of the data read from Faodel.

5 RELATED WORK

The data management services provided by Faodel can improve the efficiency, speed, and maintainability of key computational science applications. An important body of existing research has examined many similar services.

5.1 Asynchronous Many-Task Runtime Systems

Asynchronous Many-Task (AMT) runtime systems promise to eliminate many of the scaling issues that have emerged in tightly-coupled, synchronous applications by allowing independent tasks to execute as soon as their data dependencies are satisfied. Because predicting where and when a task may execute in an AMT programming model is a formidable task, AMT runtimes depend on services to provide efficient data movement. Meng et al. [11] describe how the Uintah "Data Warehouse" allows tasks to access Uintah variables. Each variables is stored in a dictionary with a well-defined key. Tasks are only allowed to work with variables stored in their local memory and variables acquired from the Data Warehouse. Faodel is aimed at providing job-to-job communication in these environments.

5.2 Data Management in Scientific Workflows

Many modern scientific simulations require multiple phases of execution to arrive at a solution. Efficient execution of these simulations depends on robust data management services for exchanging (or sharing) simulation data among the applications that comprise the workflow. Dataspaces [2] uses distributed hash tables and leverage advanced interconnect services (e.g., RDMA) to exchange data between applications (e.g., coupled physics codes). DDSS [16] is a distributed service for sharing data in a datacenter environment. The key abstraction in this approach is a logical shared memory region referenced by a unique key. Sharing data is accomplished by sharing the key associated with the data. Support for coherence and access control (e.g., granting exclusive access) are provided. Other research projects, e.g., Kepler [10] and Taverna [12], have focused on developing tools that enable domain scientists to quickly and easily assemble scientific workflows. Faodel is focusing to deliver similar capabilities to applications using other programming models such as AMT.

5.3 Data Analysis

Many scientific simulations have the potential to generate vast quantities of output data. Domain scientists rely on sophisticated analysis and visualization to make sense of these data. Efficient use of these tools requires robust data management services to find and access output datasets. Pavlo et al. [14] compare the use of MapReduce and Parallel Database Management Systems (DBMS) for analyzing large volumes of data. For both of these approaches, data is stored and exchanged through the filesystem. SENSEI [1] defines a generic data model to facilitate the transfer of data between simulation and analysis tasks. Their generic data model is intended to simplify the process of combining a simulation code with different kinds of analysis.

5.4 Unified Storage Hierarchy

HPC systems have recently experienced significant growth in the depth of their storage hierarchy. SSDs, NVRAM, and 3D-stacked DRAM are all becoming increasingly common. Because each level of the hierarchy represents a different set of tradeoffs (and possibly also, different programming interfaces), application developers face an increasingly complex set of choices when decided where their data should reside. UNITY [6] provides a single interface for applications to access all levels of storage hierarchy. Data Elevator [3] provides a transparent mechanism for moving data among different layers of the storage hierarchy. Specifically, the authors describe and demonstrate their approach to moving data between burst buffers and the parallel file system.

6 CONCLUSION

This paper describes our work to date on Faodel, a set of services we envision as a data backplane for applications in computational science workflows. Using Faodel, applications in workflows will be able to exchange data without necessarily involving a file system, while structuring those exchanges according to semantics which are meaningful to the developers involved. We have demonstrated the generality and efficiency of the Faodel software stack in several application use cases, two of which we have presented here. Our initial evaluations have been intended to demonstrate feasibility for these examples. Our future plans for Faodel include evaluations in more complex workflows with larger computational and data transfer requirements. We are also investigating integrations with workflow definition and deployment tools under development by other researchers at Sandia.

ACKNOWLEDGMENTS

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International,

Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

REFERENCES

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

[2] 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*. ACM, 25–36.

[3] Bin Dong, Suren Byna, Kesheng Wu, Hans Johansen, Jeffrey N Johnson, Noel Keen, et al. 2016. Data Elevator: Low-Contention Data Movement in Hierarchical Storage System. In *High Performance Computing (HiPC), 2016 IEEE 23rd International Conference on*. IEEE, 152–161.

[4] Richard L Graham, Timothy S Woodall, and Jeffrey M Squyres. 2005. Open MPI: A flexible high performance MPI. In *International Conference on Parallel Processing and Applied Mathematics*. Springer, 228–239.

[5] Kitware Inc. 2018. VTK - The Visualization Toolkit. http://www.vtk.org. Accessed: 2018-04-03.

[6] Terry Jones, Michael J Brim, Geoffroy Vallee, Benjamin Mayer, Aaron Welch, Tonglin Li, Michael Lang, Latchesar Ionkov, Douglas Otstott, Ada Gavrilovska, et al. 2017. UNITY: Unified Memory and File Space. In *Proceedings of the 7th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS 2017)*. ACM, 6.

[7] Margaret Lawson, Craig Ulmer, Shyamali Mukherjee, Gary Templet, Jay Lofstead, Scott Levy, Patrick Widener, and Todd Kordenbrock. 2017. Empress: Extensible Metadata Provider for Extreme-scale Scientific Simulations. In *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS '17)*. ACM, New York, NY, USA, 19–24. https://doi.org/10.1145\/3149393.3149403

[8] Jiuxing Liu, Jiesheng Wu, and Dhabaleswar K Panda. 2004. High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 32, 3 (2004), 167–198.

[9] Jay Lofstead, Ron Oldfield, Todd Kordenbrock, and Charles Reiss. 2011. Extending Scalability of Collective I/O Through Nessie and Staging. In *Proceedings of the 6th Parallel Data Storage Workshop (PDSW '11)*. Seattle, WA, 7–12. https://doi.org/10.1145/2159352.2159355

[10] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A Lee, Jing Tao, and Yang Zhao. 2006. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience* 18, 10 (2006), 1039–1065.

[11] Qingyu Meng, Justin Luitjens, and Martin Berzins. 2010. Dynamic task scheduling for the uintah framework. In *Many-Task Computing on Grids and Supercomputers (MTAGS), 2010 IEEE Workshop on*. IEEE, 1–10.

[12] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R Pocock, Anil Wipat, et al. 2004. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics* 20, 17 (2004), 3045–3054.

[13] Ron A. Oldfield, Patrick Widener, Arthur B. Maccabe, Lee Ward, and Todd Kordenbrock. 2006. Efficient Data-Movement for Lightweight I/O. In *Proceedings of the 2006 International Workshop on High Performance I/O Techniques and Deployment of Very Large Scale I/O Systems*. Barcelona, Spain. https://doi.org/10.1109/CLUSTR.2006.311897

[14] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J Abadi, David J DeWitt, Samuel Madden, and Michael Stonebraker. 2009. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 165–178.

[15] Hiroshi Tezuka, Francis O'Carroll, Atsushi Hori, and Yutaka Ishikawa. 1998. Pin-down cache: A virtual memory management technique for zero-copy communication. In *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing(IPPS/SPDP) 1998*. IEEE, 308–314.

[16] Karthikeyan Vaidyanathan, Sundeep Narravula, and Dhabaleswar K Panda. 2006. DDSS: a low-overhead distributed data sharing substrate for cluster-based datacenters over modern interconnects. *Lecture notes in computer science* 4297 (2006), 472.