Examples of *In Transit* Visualization

Kenneth Moreland
Sandia National Laboratories
kmorel@sandia.gov

Ron Oldfield
Sandia National Laboratories
raoldfi@sandia.gov

Pat Marion
Kitware, Inc.
pat.marion@kitware.com

Sebastien Jourdain
Kitware, Inc.
sebastien.jourdain@kitware.com

Norbert Podhorszki
Oak Ridge National Laboratory
pnorbert@ornl.gov

Venkatram Vishwanath
Argonne National Laboratory
venkatv@mcs.anl.gov

Nathan Fabian
Sandia National Laboratories
ndfabia@sandia.gov

Ciprian Docan
Rutgers University
docan@caip.rutgers.edu

Manish Parashar
Rutgers University
parashar@rutgers.edu

Mark Hereld
Argonne National Laboratory
hereld@mcs.anl.gov

Michael E. Papka
Argonne National Laboratory
papka@anl.gov

Scott Klasky
Oak Ridge National Laboratory
klasky@ornl.gov

ABSTRACT

One of the most pressing issues with petascale analysis is the transport of simulation results data to a meaningful analysis. Traditional workflow prescribes storing the simulation results to disk and later retrieving them for analysis and visualization. However, at petascale this storage of the full results is prohibitive. A solution to this problem is to run the analysis and visualization concurrently with the simulation and bypass the storage of the full results. One mechanism for doing so is *in transit* visualization in which analysis and visualization is run on I/O nodes that receive the full simulation results but write information from analysis or provide run-time visualization. This paper describes the work in progress for three *in transit* visualization solutions, each using a different transport mechanism.

Categories and Subject Descriptors

I.6.6 [**Computing Methodologies**]: Simulation and Modeling—*Simulation Output Analysis*

Keywords

in situ, in transit, staging, parallel scientific visualization

1. INTRODUCTION

In situ visualization refers to running a simulation concurrently with the visualization of its results. The concept of running a visualization while the simulation is running is not new. It is mentioned in the 1987 National Science Foundation Visualization in Scientific Computing workshop report [18], which is often attributed to launching the field of scientific visualization. However, the interest in *in situ* visualization has grown significantly in recent years due to recent problems in the standard simulation-visualization workflow.

Recent studies show that the cost of dedicated interactive visualization computers for petascale is prohibitive [5] and that the time spent in writing data to and reading data from disk storage is beginning to dominate the time spent in both the simulation and the visualization [26, 27, 30]. Consequently, *in situ* visualization is one of the most important research topics in large-scale visualization today [2, 13].

In transit visualization (also known as *staged visualization*) is a particularly elegant form of *in situ* visualization that exploits an I/O transport infrastructure that address the disk transfer limitations of modern supercomputers. A modern supercomputer's compute rate far exceeds its disk transfer rate. Recent studies show that the latency of the disk storage can be hidden by having a "staging" job running separately but concurrently with the main computation job. This staging job is able to buffer data and write it to disk while the main job continues to compute [1, 20, 21, 29]. Rather than dump the results straight to disk, studies show it is feasible to instead (or in addition) perform "in transit" analysis and visualization on these staging nodes as demonstrated in Figure 1.

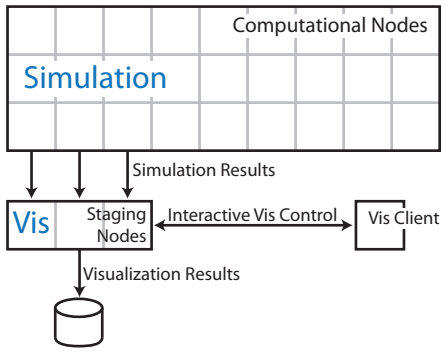

Figure 1: *In transit* **visualization leverages an I/O transport layer to intercept data and perform analysis.**

PDAC'11, November 14, 2011, Seattle, Washington, USA.

In transit visualization requires the ability to transfer data from the scientific code to the "staging" area for analysis. In techniques such as I/O Delegation [20] the applications use MPI to communicate this data. For I/O Delegation, the user allocates an additional set of staging processors when the application is launched. Then, a separate MPI communicator allows the staging processors to perform analysis without interfering with the primary application. This approach was first demonstrated for high-performance computing in a seismic imaging application called Salvo [25]. In Salvo, the user allocated an "I/O Partition" for staging outgoing data and also performing proprocessing (i.e., FFTs) on incoming data. I/O delegation is perhaps the most portable approach for *in transit* computation, but it requires a tight coupling of analysis with application and it is impossible to share the service with multiple applications.

A second more flexible approach for *in transit* visualization is to create the staging area as a separate application (or service) that communicates with the client application through a low-level network transport. This approach is extremely flexible because it allows for the potential "chaining" of application services, coupling of applications, and application sharing. The three projects described in this paper use this more loosely coupled approach.

This paper presents the work in progress for three projects performing *in transit* visualization. Each project uses a different I/O transport mechanism: the Network Scalable Service Interface (Nessie) [23], the GLEAN framework [35], and the Adaptable IO System (ADIOS) [16]. Each project demonstrates the integration of visualization with a different type of simulation. All three projects make use of the ParaView parallel visualization services [32].

2. RELATED WORK

There exist several systems designed to directly integrate simulation with visualization such as SCIRun [12], pV3 [11], and RVSLIB [8]. Other work focuses on integrating simulation codes with end user visualization tools such as ParaView [10] and VisIt [37].

These solutions require programmers to directly integrate the simulation with a visualization solution. One of the goals of *in transit* visualization is to more loosely couple these two units so that they may be applied to multiple instances without further programming. Tools such as ESPN [9] and ITAPS [4] attempt to provide more general interfaces between data producers and consumers.

In addition to those discussed here, other projects are also considering *in transit* visualization. For example, another approach leverages the XDMF/HDF5 layer as a transport mechanism for visualization [3].

3. NESSIE

The NEtwork Scalable Service Interface (Nessie) is a framework for developing *in transit* analysis capabilities [23]. It provides a remote-procedure call (RPC) abstraction that allows the application-developer to create custom data services to match the specific needs of the application.

Like Sun RPC [19], Nessie relies on client and server stub functions to encode/decode (i.e., *marshal*) procedure call parameters to/from a machine-independent format. This approach is portable because it allows access to services on heterogeneous systems, but it is not efficient for I/O requests containing raw buffers that do not need encoding. To address this marshalling issue, Nessie uses separate communication channels for control and data messages. A *control message* is typically small, identifying the operation to perform, where to get arguments, the structure of the arguments, and so forth. In contrast, a *data message* is typically large and consists of "raw" bytes that, in most cases, do not need to be encoded/decoded by the server.

To push control messages to the servers, the Nessie client uses the RPC-like interface. However, to push or pull data to/from the client, the server uses a one-sided API that accesses the system's native remote direct-memory (RDMA) capabilities. This server-directed protocol allows interactions with heterogeneous servers, but also benefits from allowing the server to control the transport of bulk data [15, 31]. The server can thus manage large volumes of requests with minimal resource requirements. Furthermore, since servers are expected to be a critical bottleneck in the system, a server-directed approach allows the server to optimize the processing of requests for efficient use of underlying network and storage devices — for example, re-ordering requests to a storage device [15].

Nessie is designed specifically for HPC systems that support RDMA and has ports for Portals, InfiniBand, Gemini, and LUC. Nessie has been used to implement services for file systems [22], HPC proxies for database access [24], and data staging for PnetCDF [29]. Ongoing work using Nessie for *in transit* analysis of the CTH shock physics code [14] is described further below.

Rather than require applications to modify code to support Nessie, a typical service developer uses the RPC framework to develop link-time replacements for libraries already in use by the application. This is the approach taken for the PnetCDF staging service, the SQL proxy, and the CTH fragment-detection service. In the case of CTH, we implement client and server stubs for the ParaView Coprocessing library [10] — an API for performing *in situ* analysis using ParaView. Instead of performing the analysis on the CTH compute nodes, our Nessie client marshals requests, sends data to the staging nodes, and performs the analysis on the staging nodes. Figure 2 illustrates this process. This approach allows fragment detection to execute in parallel with CTH, unlike a tightly coupled *in situ* approach that requires CTH to wait for the analysis to complete. This approach requires no code changes on the part of the CTH developer and it allows performance analysis comparing *in situ* verses *in transit* approaches. This performance study is ongoing and will be reported in future work.

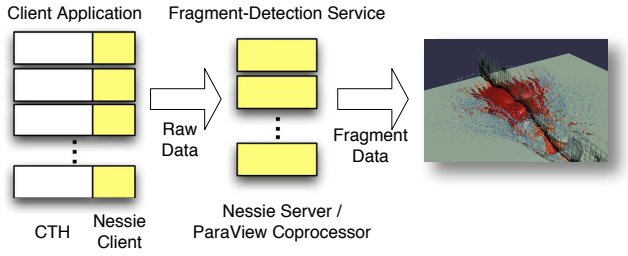

Figure 2: *In transit* **fragment detection for the CTH shock physics code.**

4. GLEAN

GLEAN is a flexible and extensible framework that takes into account application, analysis and system characteristics in order to facilitate simulation-time data analysis and I/O acceleration [35]. It is developed by the Mathematics and Computer Science Division (MCS) and Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory. To facilitate *in transit* visualization, GLEAN uses a client/server architecture to move data out of the simulation application (client) and onto dedicated staging nodes (server). The GLEAN client runs on compute nodes or on dedicated I/O nodes. It takes data I/O streams from a running solver and forwards the data to a GLEAN server. The GLEAN server runs on staging or visualization nodes that are connected to the supercomputer via a local network.

GLEAN is used as the data transport method in covis and *in situ* experiments using the PHASTA flow solver and ParaView coprocessor on an IBM BlueGene/P supercomputer. PHASTA is a parallel, hierarchic (2nd-5th order accurate), adaptive, stabilized (finite element) transient, incompressible and compressible flow solver [36]. The ParaView coprocessor is a library that provides ParaView's parallel services to a solver by linking directly with the solver binary targeted to run on compute nodes [10]. At the end of a timestep or iteration, the solver makes a function call to pass the current solution state to the ParaView coprocessor. The coprocessor reads instructions from a Python script to build a filter pipeline for *in situ* analysis of the solution data. The filter pipeline extracts meaningful information from the input data and saves the results using I/O. In this experiment, GLEAN is used as the I/O framework instead, removing the need to write to the hard disk.

The GLEAN integration with the ParaView coprocessor is implemented with a pair of Visualization Toolkit (VTK) reader and writers. (VTK is the visualization library on which ParaView is built.) To perform standard disk I/O, the user connects a geometry writer to the end of the coprocessor filter pipeline. In this experiment, we replace a standard VTK writer with the GLEAN writer. The GLEAN writer acts as a GLEAN client to re-route data to a listening GLEAN server on staging nodes. Once the data has been moved to the staging nodes, a GLEAN filter re-indexes the element arrays to account for the aggregation of the data from a large number of compute nodes to a smaller number of stage nodes. On the staging nodes, a GLEAN server combined with a standard ParaView server receives the data. The VTK GLEAN reader on the ParaView server takes the data from the GLEAN server and makes it available to the user interacting with the ParaView server.

Conversion of VTK data objects produced by the ParaView coprocessor to GLEAN transit buffers does not require copying of memory. Once the data has been moved to the staging nodes, a GLEAN filter re-indexes the element arrays to account for the aggregation of the data from a large number of compute nodes to a smaller number of stage nodes. The GLEAN reader requests arrays from the GLEAN server and re-packs them into VTK data objects, without copying the data, to serve to the consuming filters of the ParaView server.

We conducted *in transit* visualization experiments on Intrepid, an IBM BlueGene/P at ALCF, with 160,000 cores, and the Eureka visualization cluster with 400 cores and 200 GPUs. A ParaView server with coupled GLEAN server ran on Eureka, and the PHASTA solver coupled with the ParaView coprocessor ran on Intrepid. The demonstration problem simulated flow control over a full 3D swept wing as shown in Figure 3. Synthetic jets on the wing pulsed at 1750Hz, producing unsteady cross flow that increase or decrease the lift, or even reattach a separated flow.

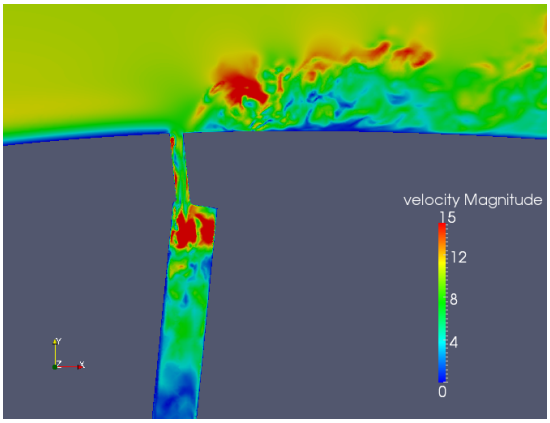

Figure 3: Cut plane through the synthetic jet simulated with PHASTA on Intrepid and visualized concurrently on Eureka.

Runs used meshes of size 22M, 416M, and 3.3B elements. At full scale, the experiment used the total amount of available cores on both systems, while a ParaView GUI connected to the ParaView server on Eureka interacted with the solver data offloaded using GLEAN transport from Intrepid and staged on Eureka.

5. ADIOS

The Adaptable I/O System framework (ADIOS) [16] is a next-generation I/O framework, which provides innovative solutions to a variety of I/O challenges facing large-scale scientific applications. ADIOS is designed to separate the I/O API from the actual implementation of the I/O methods. This design specification enables the users to easily, and without any application source code modifications, select I/O methods that are optimized for performance and functionality on the target platform. ADIOS also includes a new self-describing file format, which has shown scalability at leadership scale (> 100K cores) [17] and high consistent throughput for both writing and reading [28, 34]. Due to its componentized architecture, many institutions have contributed to the development of ADIOS methods. In fact, applications utilizing ADIOS have received over 24% of the allocated time at Oak Ridge Leadership Computing Facility.

By decoupling the APIs from the implementation, ADIOS also enables output in a variety of formats ranging from ASCII to parallel HDF5, as well as allowing the usage of new data staging techniques [6,34] that can bypass the storage system altogether. In ADIOS, the user is provided the flexibility of selecting the I/O technique through a single change in an application specific XML file thus allowing an easy transition from file-based coupling to in-memory coupling [6, 38].

This external XML file is also used to describe the data and additionally can be used to add annotations that can aid data processing downstream from the application, especially

during *in transit* visualization. For example, users can add extra information to the XML file to describe the schema for visualization.

ADIOS uses the DataSpaces method [6] to provide the abstraction of a virtual semantically specialized shared space that can be associatively and asynchronously accessed using simple, yet powerful and flexible, operators (e.g., put() and get()) with appropriate data selectors. These operators are location and distribution agnostic, allowing the *in transit* visualization to reference data without explicitly dealing with discovery and transfer. Additionally, the DataSpaces staging area exists as a separate application (see Figure 1), providing fault isolation for the application. Thus, failures in the coupled codes do not have to propagate to the application. DataSpaces can also hold multiple versions of a named dataset, for example, multiple timesteps from an application. DataSpaces also manages the available buffering in the staging area autonomously by evicting the oldest versions. This eviction policy is particularly apt for typical visualization scenarios where the data generation rate from the application needs to be decoupled from the data consumption rate of visualization.

Applications already utilizing the ADIOS API can immediately use the DataSpaces method for *in transit* visualization. DataSpaces and ADIOS allow reader applications (e.g. ParaView server or a coupled application [7]) to retrieve an arbitrary portion of a dataset; however, in order to address the unavailability of some timesteps we have had to modify the read semantics for the reading application. Data sets are referenced by a timestep and only a single timestep can be retrieved at a time. Additionally, new error codes from the file-open operation indicate whether the requested timestep is still available, whether newer steps are available, or whether a timestep will never be available because the producer has terminated.

Our *in transit* visualization application (see Figure 4) involves five separate applications.

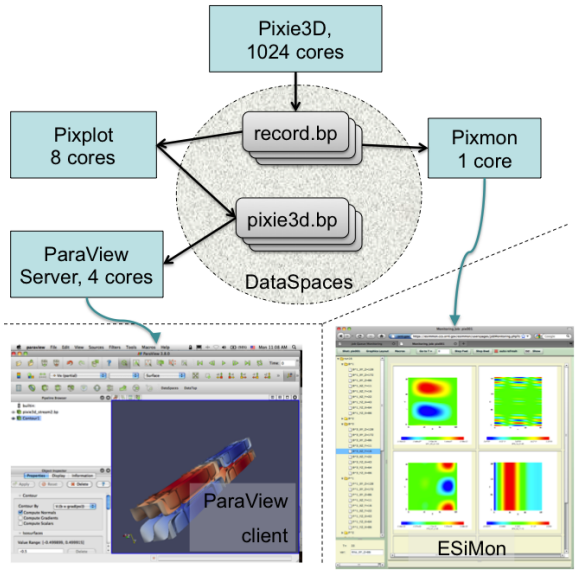

Figure 4: Using ADIOS/DataSpaces for *in transit* analysis and visualization

Pixie3D is an MHD code for plasma fusion simulation.
Pixplot is a parallel analysis code for Pixie3D output that creates a larger dataset suitable for visualization.
Pixmon creates 2D slices of the 3D output of Pixie3D and to presents them through ESiMon [33] to monitor the run.
ParaView server (with an ADIOS reader plugin) can read either from a file or from a staging area.
DataSpaces serves these four applications.

The visualization server is controlled by a ParaView client; therefore the retrieval rate of individual timesteps is varying. In our actual simulation run, Pixie3D generated about 100MB of data every second while Pixplot processed every 30th step and wrote about 500MB every 30 seconds. One compute node for staging was enough to hold 20 timesteps (generated in 10 minutes) of Pixplot result data at once and to comfortably analyze the run with ParaView. Since DataSpaces can scale to hundreds of nodes and provides both low latency and high bandwidth for data exchange, it can store all timesteps of a run of this nature if needed.

6. CONCLUSION

An *in situ* visualization system requires flexibility if it is to be applied to multiple problem domains, and we find the *in transit* approach provides a convenient mechanism to loosely couple simulation and visualization components. As noted in this paper, we are pursuing the use of the parallel ParaView server with several different I/O transport mechanisms and simulations. This work will simplify the creation of *in situ* services in simulation runs.

7. ACKNOWLEDGMENTS

Funding for this work was provided by the SciDAC Institute for Ultrascale Visualization and by the Advanced Simulation and Computing Program of the National Nuclear Security Administration.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

8. ADDITIONAL AUTHORS

9. REFERENCES

[1] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. DataStager: Scalable data staging services for petascale applications. In *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing (HPDC '09)*, 2009. DOI=10.1145/1551609.1551618.

[2] S. Ahern, A. Shoshani, K.-L. Ma, et al. Scientific discovery at the exascale. Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and Visualization, February 2011.

[3] 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 *Eurographics Symposium on Parallel Graphics and Visualization*, pages 91–100, 2011. DOI=10.2312/EGPGV/EGPGV11/091-100.

[4] K. Chand, B. Fix, T. Dahlgren, L. F. Diachin, X. Li, C. Ollivier-Gooch, E. S. Seol, M. S. Shephard, T. Tautges, and H. Trease. The ITAPS iMesh interface. Technical Report Version 0.7, U. S. Department of Energy: Science Discovery through Advanced Computing (SciDAC), 2007.

[5] H. Childs. Architectural challenges and solutions for petascale postprocessing. *Journal of Physics: Conference Series*, 78(012012), 2007. DOI=10.1088/1742-6596/78/1/012012.

[6] C. Docan, M. Parashar, and S. Klasky. DataSpaces: An interaction and coordination framework for coupled simulation workflows. In *19th ACM International Symposium on High Performance and Distributed Computing (HPDC'10)*, Chicago, IL, June 2010.

[7] C. Docan, F. Zhang, M. Parashar, J. Cummings, N. Podhorszki, and S. Klasky. Experiments with memory-to-memory coupling for end-to-end fusion simulation workflows. In *10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'10)*, pages 293–301, Melbourne, Australia, May 2010.

[8] S. Doi, T. Takei, and H. Matsumoto. Experiences in large-scale volume data visualization with RVSLIB. *Computer Graphics*, 35(2), May 2001.

[9] A. Esnard, N. Richart, and O. Coulaud. A steering environment for online parallel visualization of legacy parallel simulations. In *Proceedings of the 10th International Symposium on Distributed Simulation and Real-Time Applications (DS-RT 2006)*, pages 7–14, October 2006. DOI=10.1109/DS-RT.2006.7.

[10] N. Fabian, K. Moreland, D. Thompson, A. C. Bauer, P. Marion, B. Geveci, M. Rasquin, and K. E. 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*, October 2011.

[11] R. Haimes and D. E. Edwards. Visualization in a parallel processing environment. In *Proceedings of the 35th AIAA Aerospace Sciences Meeting*, number AIAA Paper 97-0348, January 1997.

[12] C. Johnson, S. G. Parker, C. Hansen, G. L. Kindlmann, and Y. Livnat. Interactive simulation and visualization. *IEEE Computer*, 32(12):59–65, December 1999. DOI=10.1109/2.809252.

[13] C. Johnson, R. Ross, et al. Visualization and knowledge discovery. Report from the DOE/ASCR Workshop on Visual Analysis and Data Exploration at Extreme Scale, October 2007.

[14] E. S. H. Jr., R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. McGlaun, S. V. Petney, S. A. Silling, P. A. Taylor, and L. Yarrington. CTH: A software family for multi-dimensional shock physics analysis. In R. Brun and L. Dumitrescu, editors, *Proceedings of the 19th International Symposium on Shock Physics*, volume 1, pages 377–382, Marseille, France, July 1993.

[15] D. Kotz. Disk-directed I/O for MIMD multiprocessors. In H. Jin, T. Cortes, and R. Buyya, editors, *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, chapter 35, pages 513–535. IEEE Computer Society Press and John Wiley & Sons, 2001.

[16] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Adaptable, metadata rich IO methods for portable high performance IO. In *IEEE International Symposium on Parallel & Distributed Processing, IPDPS'09*, May 2009. DOI=10.1109/IPDPS.2009.5161052.

[17] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf. Managing variability in the IO performance of petascale storage systems. In *Proceedings of the Conference on High Performance Computing, Networking, Storage and Analysis, SC'10*, New Orleans, LA, November 2010.

[18] B. H. McCormick, T. A. DeFanti, and M. D. Brown, editors. *Visualization in Scientific Computing (special issue of Computer Graphics)*, volume 21. ACM, 1987.

[19] S. Microsystems. RPC: remote procedure call protocol specification, version 2. Technical Report RFC 1057, Sun Microsystems, Inc., June 1988.

[20] A. Nisar, W. keng Liao, and A. Choudhary. Scaling parallel I/O performance through I/O delegate and caching system. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, November 2008.

[21] R. A. Oldfield, S. Arunagiri, P. J. Teller, S. Seelam, R. Riesen, M. R. Varela, and P. C. Roth. Modeling the impact of checkpoints on next-generation systems. In *Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies*, San Diego, CA, September 2007.

[22] R. A. Oldfield, A. B. Maccabe, S. Arunagiri, T. Kordenbrock, R. Riesen, L. Ward, and P. Widener. Lightweight I/O for scientific applications. In *Proceedings of the IEEE International Conference on Cluster Computing*, Barcelona, Spain, Sept. 2006.

[23] R. A. Oldfield, P. Widener, A. B. Maccabe, L. Ward, and T. Kordenbrock. 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 I/O Systems*, Barcelona, Spain, Sept. 2006.

[24] R. A. Oldfield, A. Wilson, G. Davidson, and C. Ulmer. Access to external resources using service-node proxies. In *Proceedings of the Cray User Group Meeting*, Atlanta, GA, May 2009.

[25] R. A. Oldfield, D. E. Womble, and C. C. Ober. Efficient parallel I/O in seismic imaging. *International Journal of High Performance Computing Applications*, 12(3):333–344, Fall 1998.

[26] T. Peterka, H. Yu, R. Ross, and K.-L. Ma. Parallel volume rendering on the IBM Blue Gene/P. In *Proceedings of Eurographics Parallel Graphics and Visualization Symposium 2008*, 2008.

[27] T. Peterka, H. Yu, R. Ross, K.-L. Ma, and R. Latham. End-to-end study of parallel volume rendering on the IBM Blue Gene/P. In *Proceedings of ICPP '09*, pages 566–573, September 2009. DOI=10.1109/ICPP.2009.27.

[28] M. Polte, J. Lofstead, J. Bent, G. Gibson, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, M. Wingate, and M. Wolf. ...and eat it too: High read performance in write-optimized HPC I/O middleware file formats. In *Proceedings of Petascale Data Storage Workshop 2009 at Supercomputing 2009*, November 2009.

[29] C. Reiss, G. Lofstead, and R. Oldfield. Implementation and evaluation of a staging proxy for checkpoint I/O. Technical report, Sandia National Laboratories, Albuquerque, NM, August 2008.

[30] R. B. Ross, T. Peterka, H.-W. Shen, Y. Hong, K.-L. Ma, H. Yu, and K. Moreland. Visualization and parallel I/O at extreme scale. *Journal of Physics: Conference Series*, 125(012099), 2008. DOI=10.1088/1742-6596/125/1/012099.

[31] K. E. Seamons and M. Winslett. Multidimensional array I/O in Panda 1.0. *Journal of Supercomputing*, 10(2):191–211, 1996.

[32] A. H. Squillacote. *The ParaView Guide: A Parallel Visualization Application*. Kitware Inc., 2007. ISBN 1-930934-21-1.

[33] R. Tchoua, S. Klasky, N. Podhorszki, B. Grimm, A. Khan, E. Santos, C. Silva, P. Mouallem, and M. Vouk. Collaborative monitoring and analysis for simulation scientists. In *2010 International Symposium on Collaborative Technologies and Systems, (CTS 2010)*, pages 235–244, Chicago, IL, USA, May 2010.

[34] Y. Tian, S. Klasky, H. Abbasi, J. Lofstead, R. Grout, N. Podhorszki, Q. Liu, Y. Wang, and W. Yu. Edo: Improving read performance for scientific applications through elastic data organization. In *IEEE Cluster 2011*, Austin, TX, 2011.

[35] V. Vishwanath, M. Hereld, V. Morozov, and M. E. Papka. Topology-aware data movement and staging for I/O acceleration on BlueGene/P supercomputing systems. In *IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2011)*, November 2011.

[36] C. H. Whiting and K. E. Jansen. A stabilized finite element method for the incompressible Navier–Stokes equations using a hierarchical basis. *International Journal for Numerical Methods in Fluids*, 35(1):93–116, January 2001.

[37] B. Whitlock. Getting data into VisIt. Technical Report LLNL-SM-446033, Lawrence Livermore National Laboratory, July 2010.

[38] F. Zhang, C. Docan, M. Parashar, and S. Klasky. Enabling multi-physics coupled simulations within the PGAS programming framework. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 84–93, 2011.