

Bridging HPC and Grid File I/O with IOFSL

Jason Cope¹, Kamil Iskra¹, Dries Kimpe^{1,2}, and Robert Ross¹

¹ Mathematics and Computer Science Division, Argonne National Laboratory

² Computation Institute, University of Chicago

{copej,iskra,dkimpe,ross}@mcs.anl.gov

Abstract. Traditionally there has been little interaction between the Grid and High-Performance Computing (HPC) storage research communities. Grid research often focused on optimizing data accesses for high-latency, wide-area networks while HPC research focused on optimizing data accesses for local, high-performance storage systems. Recent software and hardware trends are blurring the distinction between Grids and HPC. In this paper, we investigate the use of I/O forwarding – a well established technique in leadership-class HPC machines – in a Grid context. We show that the problems that triggered the introduction of I/O forwarding for HPC systems also apply to contemporary Grid computing environments. We present the design of our I/O forwarding infrastructure for Grid computing environments. Finally, we discuss the advantages our infrastructure provides for Grids, such as simplified application data management in heterogeneous computing environments and support for multiple application I/O interfaces.

1 Introduction

Grid computing environments, such as the National Science Foundation (NSF) funded TeraGrid project, have recently begun deploying massively-parallel computing platforms similar to those in traditional HPC centers. While these systems do not support distributed or multi-resource MPI applications[8, 2], they do support a variety of HPC applications well suited for tightly-coupled resources, including high-throughput workloads [19] and massively-parallel workloads [6]. To efficiently connect these resources, TeraGrid has focused on enhancing Grid data services. This trend is evident in the goals for the emerging third phase of TeraGrid operations, known as TeraGrid “eXtreme Digital” (XD).

This shift in resource usage and deployments aligns Grids more closely with traditional HPC data-centers, such as the DOE leadership computing facilities at Argonne National Laboratory and Oak Ridge National Laboratory. This realignment poses several data access challenges. One such challenge is enabling efficient, remote data access by Grid applications using large numbers of processing elements. Massively-parallel applications can overwhelm file systems with large numbers of concurrent I/O requests. Leadership-class computing platforms face a similar data access problem for local data access to high-performance storage systems. Grid computing platforms experience

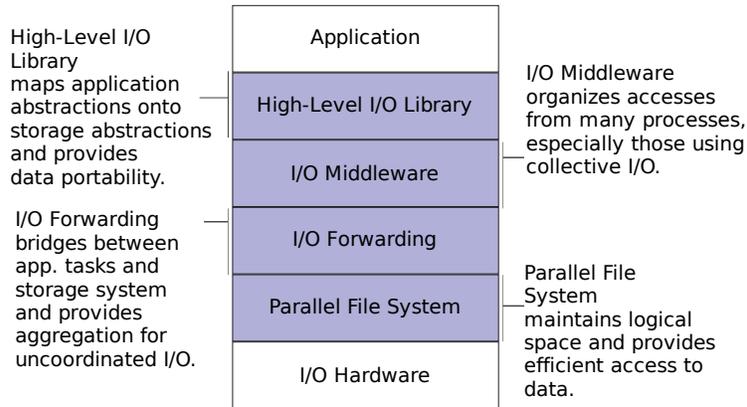


Fig. 1. I/O Forwarding in HPC systems

similar problems for both local and remote data accesses. Existing Grid data management tools do not address the impact of increased concurrency on remote data access performance or account for the limited capacity of network and storage resources as application data continues to increase.

In this paper, we describe how I/O forwarding can improve the performance of Grid application data accesses to both local and remote storage systems. In the following sections, we present our I/O forwarding infrastructure for Grid computing environments and how this infrastructure optimizes application remote data accesses in Grids. Section 2 presents I/O forwarding and its use in HPC. Section 3 describes typical I/O mechanisms used by Grid applications and how I/O forwarding integrates into Grids. In Section 5, we describe related work and conclude this paper.

2 HPC I/O

In this section, we introduce the concept of I/O forwarding, followed by a description of our portable, open source implementation.

2.1 A Revised I/O Software Stack

The current generation of leadership-class HPC machines such as the IBM Blue Gene/P supercomputer at Argonne National Laboratory or the *Roadrunner* machine at the Los Alamos National Laboratory consist of a few hundred thousand processing elements. Future generations of supercomputers will incorporate millions of processing elements. This significant increase in scale is brought about by an addition in the number of nodes along with new multi-core architectures that can accommodate an increasing number of processing cores on a single chip.

While the computation power of supercomputers keeps increasing with every generation, the same is not true for their I/O subsystems. The data access

rates of storage devices has not kept pace with the exponential growth in processing performance. In addition to the growing bandwidth gap, the increase in compute node concurrency has revealed another problem: the parallel file systems available on current leadership-class machines, such as PVFS2 [4], GPFS [14], Lustre [5] and PanFS [12] were designed for smaller systems with fewer file system clients. While some of these file systems incorporate features for enhanced scalability, they are often not prepared to deal with the enormous increase in clients brought on by the increasing trend towards more concurrency.

MPI-IO, distributed as part of the MPI library, is the standard parallel I/O API for HPC systems. In certain cases, by using *collective I/O*, the MPI-IO implementation is able to reduce the number of requests made to the filesystem. However, not all applications use the MPI-IO interface or are able to use collective I/O, so improvements made at the MPI-IO layer may not be available to the entire spectrum of scientific applications. Parallel high-level libraries such as Parallel-NetCDF [11] use MPI-IO and as such face many of the same limitations outlined above. POSIX implementations and serial high-level libraries are an artifact from an earlier generation and are only available on current HPC systems to support legacy applications.

To address this I/O bottleneck, another layer needed to be introduced into the I/O software stack. Clients, instead of directly making requests to the parallel filesystem, forward their I/O operations to an I/O forwarder node, which performs the I/O operation on their behalf. One I/O node is typically responsible for 32 to 128 compute clients. Due to its position in the I/O path, the I/O forwarder is able to perform a wide range of optimizations that were not previously possible. For example, it can aggregate requests of unrelated software running on multiple compute nodes. As such, it reduces both the number of requests and the number of clients visible to the parallel filesystem. Since the I/O forwarding software – running on the I/O node – does not share any resources (CPU or memory) with the compute clients, it is free to dedicate memory and compute power to optimizing I/O traffic without slowing down computation.

Another benefit of moving the actual I/O calls to the forwarder is that the compute client can be simplified. Instead of requiring a full I/O stack, it only needs to be able to send and receive requests to the I/O forwarder. The I/O forwarder then takes care of using the correct protocol to access the remote filesystem. Likewise, authentication (to the remote filesystem) can be handled by the I/O forwarder. This enables compute clients to use a simpler, local authentication scheme to authenticate to the I/O forwarder. Figure 1 shows the resulting I/O software stack.

2.2 I/O Forwarding Scalability Layer (IOFSL)

In view of the importance of I/O forwarding in HPC systems, it is desirable to have a high quality implementation capable of supporting multiple architectures, file systems and high-speed interconnects. While a few I/O forwarding solutions are available for the IBM Blue Gene and other leadership class platforms, such as the Cray XT, they are each tightly coupled to one architecture [20, 7]. The lack

of an open-source, high-quality implementation capable of supporting multiple architectures, file systems and high-speed interconnects has hampered research and makes the deployment of novel I/O optimizations difficult.

To address this issue, we created a scalable, unified I/O forwarding framework for high-performance computing systems called the I/O Forwarding Scalability Layer (IOFSL) [1]. IOFSL includes features such as the coalescing of I/O calls on the I/O node, reducing the number of requests to the file system, and full MPI-IO integration, which translates into improved performance for MPI applications. Ongoing work includes the integration of some other techniques for improving HPC I/O performance, such as [21] and [13].

3 Grid Data Access

Two approaches to application data accesses in Grids have emerged. They are described in section 3.1. Section 3.2 describes how IOFSL can be used to improve the performance and enhance the use-ability of these approaches.

3.1 Traditional Grid I/O

The first approach stages data at the resource where the application executes or offloads data locally generated by an application to a remote storage system. This approach often uses GridFTP to perform bulk data transfers between the high-performance storage systems attached to Grid resources. While this approach offers good performance, as remote I/O is only used for staging files in and out the local storage, it has a number of drawbacks. For one, it is hard to maintain consistency between the local and remote copy. The second issue is related to the access granularity. Typically, the whole file needs to be transferred, reducing efficiency if the application only requires a subset of the file data.

The second approach is to host data on wide-area file systems. These file-systems construct a distributed, shared storage space, which is mounted locally on each Grid resource to provide local application access. Examples of Grid specific filesystems include Gfarm [16] and Chirp [17]. These filesystems typically do not provide traditional I/O semantics and are currently not well supported by parallel applications. For example, in Gfarm, files are basically write-once and parallel read-write I/O has to be emulated through versioning and creating new files[15].

In addition to these Grid specific filesystems, traditional HPC filesystems such as Lustre and GPFS have been adapted for Grid environments. While these do offer familiar parallel I/O facilities, the high latencies and large number of filesystem clients severely limits their performance and stability.

3.2 I/O Forwarding in a Grid environment

When designing IOFSL, portability and modularity was an important goal. IOFSL does not make any assumptions about operating system kernels, inter-

connects, filesystems or machine architectures. As such, it can be easily retargeted to other environments, such as computational Grids.

In large HPC systems, I/O forwarding isolates local compute clients, connected by a high bandwidth, low latency interconnect from the more distant, higher latency parallel filesystem. At the same time, it protects the filesystem from being crippled by a storm of requests, by aggregating and merging requests before sending them to the filesystem. From the viewpoint of the remote filesystem, this reduces the number of visible clients and requests, hence increasing performance.

In a Grid environment, these optimizations are also applicable, albeit on a different scale. While latencies might be much higher, the same discontinuity exists when an application, running on a local Grid resource needs to fetch data from a remote data store. As is the case in large HPC systems, a large number of simultaneous requests to a remote site might adversely affect the stability and throughput of the remote file server. This observation is valid both for data staging and wide area Grid filesystems.

Figure 2 shows the location of I/O forwarding in a Grid environment. Being located at the gateway between the local compute resources and the remote data, IOFSL acts as both a connection and request aggregator: local applications can share the same set of outgoing connections, increasing efficiency and reducing the load on the remote filesystem. For example, if GridFTP is used as a data transport between the site where data is stored and the site where data is consumed or generated, when using IOFSL, the number GridFTP connections will not depend on the number of clients. Instead, each I/O forwarder can be configured to use an optimal number of GridFTP connections to obtain the data. Clients interacting with the I/O forwarder will transparently share existing connections when possible.

Another important advantage of deploying I/O forwarding in a Grid environment is that, to the client software, IOFSL can offer a more familiar access API. Currently, IOFSL implements two client side APIs: POSIX and MPI-IO. For POSIX, there is a FUSE and SYSIO implementation. The former enables redirecting I/O accesses of unmodified binary applications. While the latter requires relinking applications with the SYSIO library, it provides support on platforms that do not support FUSE (for example, minimal OS kernels such as Cray's Catamount kernel[9] or IBM BG/P's compute node kernel).

By directly supporting MPI-IO, the defacto I/O API for parallel MPI programs, IOFSL enables unmodified MPI applications (such as parallel analysis or visualization software) to transparently access remote data using GridFTP or other protocols not normally supported by HPC software. In this case, IOFSL effectively acts as a bridge between a local HPC programs and remote Grid-style storage.

Dedicating some nodes as I/O forwarders also helps with high latency network links, a typical problem when spanning multiple remote sites using a POSIX-like filesystem such as Lustre or GPFS. By using local system memory of the I/O forwarders for buffering read and write data, IOFSL is able to transform

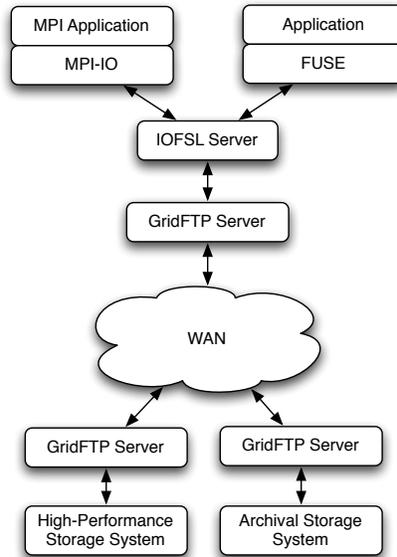


Fig. 2. I/O Forwarding in a Grid environment

synchronous client accesses into asynchronous remote accesses, reducing the detrimental effects high latency links. Often requested data can be buffered locally, where it can be accessed over a low-latency high bandwidth network. As IOFSL transparently captures all file accesses, I/O requests from multiple programs can be optimized if they are requesting the same data. For example, several independent requests to the same data can be coalesced to a single request.

3.3 IOFSL GridFTP Module Implementation

IOFSL provides access to remote data using the GridFTP driver. This driver is similar to other IOFSL drivers because it bridges generalized application I/O requests to a specific I/O subsystem. This IOFSL driver maps the GridFTP API to the ZoidFS API. The driver manages all GridFTP file handles in red-black trees and provides the application with portable, opaque IOFSL file handles. Using this driver, IOFSL servers can proxy application I/O requests to remote GridFTP servers when application cannot directly access the data because of IP routing constraints or other connection limitations. There were several challenges when implementing this driver and it contains several features that existing IOFSL drivers do not have. We currently use the GridFTP 4.2 client library to provide GridFTP support.

In order to access remote data, the location of the data must be encoded into the I/O request. For other IOFSL drivers, such as the POSIX and PVFS2 drivers, the file path is sufficient for IOFSL to locate the data since those file systems are locally available to the nodes hosting IOFSL software. To access remote data with the IOFSL GridFTP driver, we require that applications prefix the file path with the remote access protocol to use, the remote host address, and the port the GridFTP server is using. For example, an application that requires access to the `/etc/group` file hosted on server 192.168.1.100 that hosts a GridFTP server listening on port 12345 using the ftp protocol will construct a file path `/ftp/192.168.1.100/12345/etc/group`.

Unlike other IOFSL drivers, the GridFTP client uses an asynchronous operation model. The existing IOFSL drivers use synchronous data management operations and these operations are easier to adapt to the synchronous ZoidFS interface. To map the asynchronous GridFTP operations to the synchronous ZoidFS interfaces, we developed a set of callbacks and monitors that poll the GridFTP client library for operation completion. Supporting these operations also required additional locking within the ZoidFS driver operations to protect the GridFTP library from concurrent requests. Without additional optimizations, the additional locking within this driver can limit the performance of the IOFSL because of reductions in parallelism. Fortunately, higher level IOFSL optimizations that can aggregate multiple operations into a single request will reduce the number of pending GridFTP operations and lock contention with the IOFSL GridFTP driver.

The GridFTP 4.2 client library used by the IOFSL driver did not full support the ZoidFS capabilities and interface. Several operations, including link and symlink, are not available through GridFTP and IOFSL can not support these operations for applications. GridFTP cannot provide all file attributes, including file access times and group identifiers. For attribute retrieval operations, the ZoidFS GridFTP backend will fetch the available attributes and assumes that application is aware that other attributes are invalid. List I/O capabilities are supported for GridFTP writes operations, but are not supported for GridFTP read operations. The IOFSL GridFTP driver must treat all read list I/O requests as individual requests, which increases the number of requests inflight that the server must manage.

4 Evaluation

4.1 Test Setup Description

To demonstrate the basic capability of this driver, we performed several experiments to evaluate the GridFTP driver functionality and the baseline performance of the GridFTP driver compared to an existing IOFSL driver. We used the Argonne Leadership Computing Facility's Eureka Linux cluster. Eureka is a 100-node Linux cluster that uses a Myricom 10G interconnect. Each node in the cluster contains 8 Intel-based cores and 16GB of RAM. In these tests, the compute nodes of this cluster executed the application code and the login nodes

hosted our GridFTP and IOFSL servers. All network communication in these experiments use TCP/IP.

4.2 Comparison with POSIX I/O

In the following tests, we evaluated the write performance of the GridFTP driver to a local file system (accessed through a GridFTP server) on the Eureka cluster login node. We also collected data for these experiments using the POSIX IOFSL driver. The POSIX driver experiments accessed the data directly. When using the IOFSL GridFTP driver, application I/O requests are forwarded to the IOFSL server and the IOFSL server delegates the application requests to the GridFTP server.

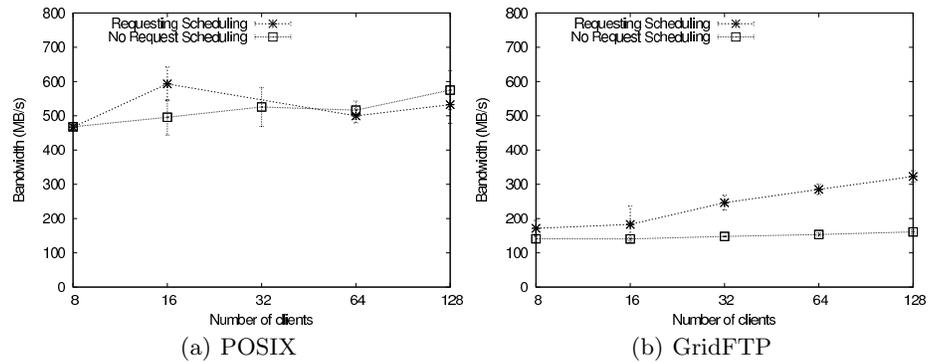


Fig. 3. Effect of request scheduling on POSIX and GridFTP access methods

From these experiments, we observed that the request merging optimization increased the performance of the GridFTP driver as the number of application processes increased. Figure 3 illustrates the results of these experiments. Without request merging, the overhead for issuing each I/O request is large due to the GridFTP server overhead and the additional locking within the IOFSL GridFTP driver. The request merging optimization is able to aggregate multiple I/O requests into a single list I/O request to the GridFTP server. For high latency network connections or I/O requests, this optimization can improved performance through the reduction of I/O requests. We observe this improvement when using the optimization with the GridFTP driver for IOFSL. For the POSIX access method, having a lower per-request cost, the effect is less clear.

Note that figure 3 is not meant to compare the performance of the GridFTP access method with that of the POSIX access method. These methods each serve distinct purposes and typically only one of them will be available for accessing a specific file. For example, while almost all compute nodes fully

support POSIX I/O, GridFTP access from a compute node will rarely be available, due to network limitations (the compute nodes do not have direct outside access) or software restrictions (microkernel operating systems limit what software portability). The primary contribution of the IOFSL GridFTP module is that it provides a remote data access capability for systems that limit remote connectivity to compute nodes or other internal infrastructure.

5 Related Work and Conclusions

5.1 Related Work

In [3], a method is described to allow MPI-IO access to GridFTP stores. It differs from our work in that the MPI application itself makes the GridFTP connection, as opposed to the I/O forwarder node when IOFSL is used. This precludes optimizations such as request merging or link aggregation.

Stork [10] tries to improve I/O access time by explicitly scheduling data staging. While IOFSL will also buffer data using local temporary storage, it does this transparently – without explicit data staging – and on a sub-file granularity.

Condor [18] enables remote I/O by shipping I/O operations back to the submission site. It requires application to relink with the condor library. While our approach also uses function call forwarding, the calls are not shipped to the remote site but to local aggregators.

5.2 Conclusions

In this paper, we provide an overview of the IOFSL project and how the I/O forwarding layer can be used to bridge HPC and Grid file I/O requests. We describe the concept of I/O forwarding in HPC systems and show how the same technique can be applied to Grid computing environments. We discuss its advantages and disadvantages, and show how it enables connecting existing HPC and posix applications with Grid data stores.

We demonstrate how our work enables transparent GridFTP access. We evaluated our initial GridFTP driver using the IOR benchmark to simulate an I/O bound application accessing remote data within a cluster. This driver demonstrates that we can effectively bridge HPC and Grid file I/O requests and service remote data requests of applications without modifications to the applications. Our current work includes improving the performance of the driver by reducing lock contention within the GridFTP driver and evaluating the use of this driver to proxy I/O requests from the compute nodes of an IBM Blue Gene/P system to remote data sources.

Acknowledgments

This work was supported by the Office of Advanced Scientific Computer Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357. The IOFSL project is supported by the DOE Office of Science and

National Nuclear Security Administration (NNSA). 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.

References

1. Ali, N., Carns, P., Iskra, K., Kimpe, D., Lang, S., Latham, R., Ross, R., Ward, L., Sadayappan, P.: Scalable I/O Forwarding Framework for High-Performance Computing Systems. In: IEEE Int'l Conference on Cluster Computing (Cluster Computing 2009) (September 2009)
2. Allen, G., Dramlitsch, T., Foster, I., Karonis, N., Ripeanu, M., Seidel, E., Toonen, B.: Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus. In: Proceedings of SC 2001 (November 10-16 2001)
3. Baer, T., Wyckoff, P.: A parallel I/O mechanism for distributed systems. In: cluster. pp. 63–69. IEEE (2004)
4. Carns, P.H., Ligon III, W.B., Ross, R.B., Thakur, R.: PVFS: A parallel file system for Linux clusters. In: Proceedings of the 4th Annual Linux Showcase and Conference. pp. 317–327 (2000)
5. Cluster File Systems, Inc.: Lustre: A scalable high-performance file system. Tech. rep., Cluster File Systems (Nov 2002), <http://www.lustre.org/docs/whitepaper.pdf>
6. Grinberg, L., Karniadakis, G.: A scalable domain decomposition method for ultra-parallel arterial flow simulations. *Communications in Computational Physics* 4(5), 1151–1169 (2008)
7. Iskra, K., Romein, J.W., Yoshii, K., Beckman, P.: ZOID: I/O-forwarding infrastructure for petascale architectures. In: ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 153–162. Salt Lake City, UT (2008)
8. Karonis, N., Toonen, B., Foster, I.: Mpich-g2: A grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing* (2003)
9. Kelly, S., Brightwell, R.: Software architecture of the light weight kernel, Catamount. In: Proceedings of the 2005 Cray User Group Annual Technical Conference (2005)
10. Kosar, T., Balman, M.: A new paradigm: Data-aware scheduling in grid computing. *Future Generation Computer Systems* 25(4), 406–413 (2009)
11. Li, J., Liao, W., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., Zingale, M.: Parallel netCDF: A high-performance scientific I/O interface. In: ACM/IEEE Conference on Supercomputing. Phoenix, AZ (Nov 2003)
12. Nagle, D., Serenyi, D., Matthews, A.: The Panasas ActiveScale storage cluster—delivering scalable high bandwidth storage. In: ACM/IEEE Conference on Supercomputing. Pittsburgh, PA (Nov 2004)
13. Nowoczynski, P., Stone, N., Yanovich, J., Sommerfield, J.: Zest: Checkpoint storage system for large supercomputers. In: 3rd Petascale Data Storage Workshop Supercomputing. pp. 1–5 (November 2008)
14. Schmuck, F., Haskin, R.: GPFS: A shared-disk file system for large computing clusters. In: USENIX Conference on File and Storage Technologies. Monterey, CA (2002)

15. Tatebe, O., Morita, Y., Matsuoka, S., Soda, N., Sekiguchi, S.: Grid datafarm architecture for petascale data intensive computing. In: ccgrid. p. 102. Published by the IEEE Computer Society (2002)
16. Tatebe, O., Soda, N., Morita, Y., Matsuoka, S., Sekiguchi, S.: Gfarm v2: A Grid file system that supports high-performance distributed and parallel data computing. In: Proceedings of the Computing in High Energy and Nuclear Physics Conference (CHEP04) (2004)
17. Thain, D., Moretti, C., Hemmes, J.: Chirp: a practical global filesystem for cluster and Grid computing. *Journal of Grid Computing* 7(1), 51–72 (2009)
18. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: The Condor experience. *Concurrency and Computation Practice and Experience* 17(2-4), 323–356 (2005)
19. Wilde, M., Ioan Raicu, I., Espinosa, A., Zhang, Z., Clifford, B., Hategan, M., Kenny, S., Iskra, K., Beckman, P., Foster, I.: Extreme-scale scripting: Opportunities for large task-parallel applications on petascale computers. *Journal of Physics: Conference Series* 180(1) (2009)
20. Yu, H., Sahoo, R.K., Howson, C., Almasi, G., Castanos, J.G., Gupta, M., Moreira, J.E., Parker, J.J., Engelsiepen, T.E., Ross, R., Thakur, R., Latham, R., Gropp, W.D.: High performance file I/O for the Blue Gene/L supercomputer. In: International Symposium on High-Performance Computer Architecture (Feb 2006)
21. Zheng, F., Abbasi, M., Docan, C., Lofstead, J., Liu, Q., Klasky, S., Prashar, M., Podhorszki, N., Schwan, K., Wolf, M.: Predata - preparatory data analytics on peta-scale machines. In: Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium (2010)

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.