Common HEC I/O Forwarding Scalability Layer
Progress Report

DOE award number:	DOE GRT00009497 / 60013581
Project title:	Common HEC I/O Forwarding Scalability Layer
Co-PI:	Ananth Devulapalli
Institution:	Ohio Supercomputer Center (OSC)
Project reporting period:	1-June-2008 to 31-May-2009
Key accomplishments:	A working prototype of I/O forwarding layer system. Publication in the peer-reviewed Cluster 2009 conference.

---

1 Introduction

Ohio Supercomputer Center (OSC) is a junior partner in the project titled *Common HEC I/O Forwarding Scalability Layer*. The goal of this project is to design and implement an *open* platform for scalable I/O forwarding for the next generation leadership class machines. These machines are going to be made up of hundreds of thousands of nodes, and current distributed file system architectures cannot scale to such large number of clients due to problems caused by large *fan-in*. One solution to that problem is to add another layer of machines between the file servers and the clients, which can intermediate the I/O requests. Not only does it reduce the fan-in problem at the file servers, but this additional layer of indirection also allows architectural flexibility, like the ability to support heterogeneous networks and file systems.

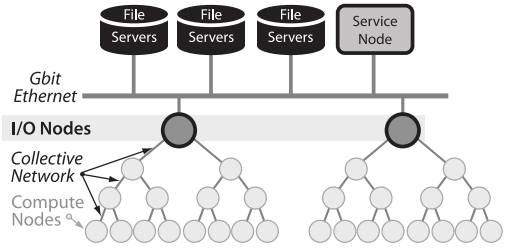

Figure 1: High level overview of IOFSL

Figure 1 shows how the I/O forwarding layer intermediates between large number of clients and file system servers. In the figure, I/O forwarding layer runs at the I/O nodes. Compute nodes dispatch their I/O requests to these I/O nodes. The I/O forwarding layer running on these nodes in turn forwards these requests to the file servers. Without these I/O nodes there will be a fan-in problem at the file servers and they will suffer from the in-cast problem [9]. Further, since all the I/O requests pass through the I/O nodes, they provide a single site for enabling support for heterogeneous file systems and interconnects.

At present the I/O forwarding layers associated with large machines like the IBM Blue Gene/P are custom designed and closely tied to the architecture of the machine on which they are implemented. The goal of this project is to design an *open* system that could be adapted for the multi-peta-flop scale machines of the near future.

During the first year of the project, the goal was to realize a prototype I/O forwarding layer and test it on diverse networks. OSC's key accomplishments during the first year include:

- Design and implementation of the I/O Forwarding Scalability Layer (IOFSL) including the client and server libraries, and their communication protocol. IOFSL is one of the key deliverables of the project.
- Implementation of the IOFSL on a portable messaging layer that enhances its portability on future leadership class machines.
- Along with other collaborators we have a publication in Cluster 2009 conference [1], related technical reports and presentations.

In the next section we give a high level overview of the design of the IOFSL. In Section 3 we give describe our accomplishments during the first year. Section 4 lists the plans for year 2009-2010. And finally we give administrative details in Section 5.

2 Design of IOFSL

In order to understand the tasks done in the first year, it is important to get a high level overview of the IOFSL. In this section we describe the IOFSL system, its components and how it is used in the cluster ecosystem. Figure 2 shows a schematic of different components of IOFSL layer. The figure shows the three entities, the client processing node, the I/O forwarding server and the system interconnect. Each entity itself is made up of a number of layers. Among the different layers, OSC was responsible for the following:

ZOIDFS client: This component runs on the compute nodes which allows them to ship their I/O requests to the intermediate I/O forwarding nodes.

ZOIDFS server: This daemon runs on the I/O nodes. It is the key component which makes I/O forwarding possible. It is responsible for parsing the I/O requests sent by the clients and translating it to corresponding requests to the appropriate file system servers. Similarly the response from the server is marshalled back to the clients in the format it understands.

Communication protocol: This is the protocol used by ZOIDFS client and servers to communicate between themselves.

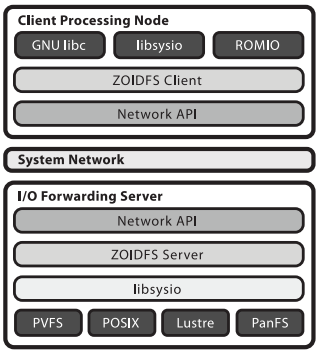

Figure 2: Components of I/O forwarding layer

IOFSL prototype implementation includes the ZOIDFS client, the ZOIDFS server and their communication protocol. However, in order for the system to be useful for the applications, appropriate interfaces must be created. For example, as the figure shows, there are several other layers in the stack besides the three components of IOFSL.

Specifically, on the client side applications use ROMIO driver to make I/O requests, which are translated into ZOIDFS API. The ZOIDFS client library supplies this API which enables it to translate client's I/O request into ZOIDFS understandable format. ROMIO driver was written by ANL.

At the ZOIDFS server, the server has the capability of multiplexing clients requests to different file systems, for example, PVFS, POSIX etc. The server uses the SYSIO library as a higher abstraction layer to achieve this purpose. As shown in Figure 2 different file systems must supply a driver which plugs under SYSIO library, so that calls can be forwarded to the appropriate file system. The drivers for different file system are being developed by our collaborators from ANL, University of Chicago and Sandia.

3 Accomplishments

In this section we give more details about the tasks accomplished during the first year. The important deliverables are the implementation of IOFSL and a publication in a peer-reviewed conference.

3.1 Implementation of IOFSL

Nawab Ali from Ohio Supercomputer Center (OSC) was the primary developer of the IOFSL project between June 2008 and June 2009. When Nawab joined the IOFSL project it was in preliminary stages. However a well-defined I/O forwarding API (ZOIDFS) had already been developed and it was being used by an earlier ZOID [6] project. Researchers at Argonne had also developed a ROMIO and PVFS driver for the ZOIDFS API to support the Blue Gene/P specific ZOID I/O forwarding implementation. During the initial meetings with the personnel at Argonne, it was decided that the ZOIDFS API will be used as a starting point for the IOFSL project. The API had been custom-designed for I/O forwarding and it would also allow us to reuse the existing ROMIO and PVFS drivers. During those early discussions, it was also decided that Nawab would focus on development of a communication API between the compute nodes and I/O nodes, implementing a custom data encoding/decoding protocol, and implementing a working prototype of the I/O forwarding framework.

Communication API: Since the I/O forwarding framework is designed to operate on multiple high-speed interconnects, we needed a communication API to abstract network communication. Buffered Message Interface (BMI) is a network abstraction layer designed for high-performance parallel I/O [3]. BMI enables parallel file systems to operate on multiple interconnection networks such as TCP/IP, InfiniBand, and Myrinet. While message-passing architectures such as Portals and MPI also provide network abstractions, BMI has inherent support for parallel I/O communication patterns. For instance, the BMI *list* operations allow applications to send or receive a set of non-contiguous buffers in a single function call. BMI was developed as part of the PVFS project [4], and until now the implementation was distributed only with PVFS. One of tasks was to decouple the BMI source code from the PVFS source tree. This task was not straight forward due to non-trivial and implicit dependencies in BMI and took us around two months to accomplish. Once BMI was separated, it enabled us to remove any dependencies on PVFS for the IOFSL project. BMI currently exists as a stand alone package and can be used independent of PVFS.

Encoding/Decoding routines: The I/O forwarding framework is designed to operate on heterogeneous machine architectures. To account for possible heterogeneity between the compute node and I/O node architectures, we encode and decode the function parameters using XDR [11]. We implemented the XDR encoding/decoding routines for all the ZOIDFS data structures. It also involved evaluation of the performance of XDR on BG/P to ensure that it would not be a bottleneck on slower processors. We also had the option of implementing custom encoding/decoding routines such as the ones used in PVFS. However, after evaluating the performance of XDR on Linux clusters and IBM BG/P, we did not see the need for custom encoding/decoding routines. XDR performs reasonably well on most modern architectures. The implementation and performance evaluation of XDR on BG/P and Linux clusters took one month of effort.

IOFSL client library: The IOFSL client library (ZOIDFS client), implements the ZOIDFS API. It runs on the compute nodes where it encodes the function parameters before shipping them to the I/O nodes over BMI. It also returns the results of the I/O operations back to the application.

IOFSL server: The IOFSL server (ZOIDFS server) is a daemon that runs on I/O nodes. It receives the encoded I/O requests from compute nodes, decodes the requests, and performs I/O on behalf of the compute nodes. The current implementation uses a pool of threads to concurrently service requests from multiple clients. Once the IOFSL server has received and decoded the I/O requests from the compute nodes, it invokes the corresponding file system driver via an I/O dispatcher. The dispatcher identifies file systems based on the ZOIDFS handle and calls the corresponding driver code. Our collaborators have developed ZOIDFS drivers for PVFS and POSIX-compliant file systems. The IOFSL server turned out to be most challenging component of the I/O forwarding framework. We faced significant issues using BMI simultaneously in client and server mode. These issues were related to bugs in the BMI code and were eventually resolved. We also evaluated multiple options in the design of the ZOIDFS POSIX driver. These have been documented in the technical report. It took us around four months to stabilize the development of IOFSL client and server libraries.

3.2 Publications and source code

IOFSL was implemented and evaluated on a Linux cluster. It is currently being ported on several machines by our collaborators. The details of the evaluation and the design are published the following paper in 2009 Cluster conference:

> Nawab Ali, Philip Carns, Kamil Iskra, Dries Kimpe, Samuel Lang, Robert Latham, Robert Ross, Lee Ward, and P. Sadayappan. Scalable I/O forwarding framework for high-performance computing systems. In *IEEE International Conference on Cluster Computing*, New Orleans, LA, August 2009.

A more detailed version of the paper is available in a related technical report [2].

IOFSL source code is publicly available from the following subversion repository:
`https://svn.mcs.anl.gov/repos/iofsl`

4 Plans for following years

The I/O forwarding layer provides a platform for optimizing the file system I/O traffic. We plan to pipeline the I/O requests between the compute nodes and I/O nodes and leverage the knowledge of larger I/O patterns to aggregate I/O requests and perform data and metadata caching. The other areas for future work include designing a capability-based security model for the ZOIDFS protocol and integrating the SYSIO library with the I/O forwarding stack. We are also developing a BMI driver for the IBM Blue Gene/P tree network. This will enable the IOFSL code to run on the Intrepid system at Argonne National Laboratory. Some other areas of future work are:

I/O pattern mining: The ZOIDFS servers are intermediaries between the real file system and clients. The advantage of these servers is the they have relevant information about I/O patterns, hot objects and metadata queries. This information can be mined for better prefetching support [5].

Improved caching: As the ZOIDFS servers get the requests at a higher object level, they are aware of objects in high demand or "hot" objects, so the servers can cache these objects aggressively. Since size of the metadata is far less than size of the data, the metadata can be aggressively cached. In several distributed file

systems, the number of the metadata requests far exceed the number of the I/O requests [10]. If the metadata at ZOIDFS servers (I/O nodes) is consistent, then several metadata requests could be trapped locally at the I/O nodes, which can improve the overall network bandwidth utilization. Further, the metadata on ZOIDFS servers can be updated in bulk by coalescing several small metadata requests into large message which amortizes the network cost over several requests. However, to keep the metadata synchronously updated in such a big system is costly. New strategies must be investigated to improve metadata performance.

Speculative execution: For stateless file systems like PVFS [4] and NFS [8], sophisticated caching techniques like callback notifications cannot be implemented. This forces them to rely on timeout based mechanisms for cache maintenance. However speculative execution [7] seems promising in improving the performance of distributed stateless file systems, by speculatively allowing the clients to execute on the tasks based on the state of the cached data, while simultaneously inquiring about the state of the cached item from the server. Speculation has been demonstrated to deliver near local file system performance. We can use speculative techniques in the intermediate ZOIDFS servers for metadata and long lived objects.

5 Administrative issues

In this section we give details of personnel involved in the project and budget expenses during first year.

5.1 Personnel changes

The three key personnel involved in the project from OSC are Dr. Pete Wyckoff, Ananth Devulapalli and Nawab Ali. Dr. Pete Wyckoff and Ananth Devulapalli are from OSC, while Nawab Ali is a PhD student from the Ohio State University, and working with the OSC staff. Dr. Wyckoff led the project for first three months from June-2008 until Aug-2008, when he left OSC. Then Ananth Devulapalli took over as Co-PI. This change was reported promptly to the DOE project officer by the OSC business officer. Nawab Ali, who was responsible for the project from OSC, was already working with our collaborators from Argonne National Laboratory (ANL) during the summer of 2008. Therefore personnel changes did not impact the progress of the project.

Item	**Expenditure** ($)
Salaries	32,367.26
Benefits	5,263.83
Tuition	2,114.00
Space Rent	468.84
F&A	9,784.07
TOTAL	49,998.00

Table 1: Summary of expenditure under different budget heads

5.2 Budget

The budget for the year June-2008 to May-2009 was $49,998. Table 1 gives a summary of how the budget was spent at OSC. In the table F&A stands for facilities and administrative overhead which is 26% of salaries and benefits.

Personnel	Salary ($)	Benefits ($)
Nawab Ali	21,687.27	1,409.67
Pete Wyckoff	6,603.69	2,366.32
Ananth Devulapalli	4,076.30	1,487.84

Table 2: Salary and benefits for each personnel

Table 2 shows the breakdown of the salaries and benefits on each personnel. As discussed in Section 5.1, Nawab was mainly responsible for implementation and most of the budget went towards supporting him.

References

[1] Nawab Ali, Philip Carns, Kamil Iskra, Dries Kimpe, Samuel Lang, Robert Latham, Robert Ross, Lee Ward, and P. Sadayappan. Scalable I/O forwarding framework for high-performance computing systems. In *IEEE International Conference on Cluster Computing*, New Orleans, LA, August 2009.

[2] Nawab Ali, Philip Carns, Kamil Iskra, Dries Kimpe, Samuel Lang, Robert Latham, Robert Ross, Lee Ward, and P. Sadayappan. Scalable I/O forwarding framework for high-performance computing systems. Technical Report OSU-CISRC-4/09-TR07, The Ohio State University, 2009.

[3] Philip H. Carns, Walter B. Ligon III, Robert Ross, and Pete Wyckoff. BMI: A network abstraction layer for parallel I/O. In *IEEE International Parallel and Distributed Processing Symposium, Workshop on Communication Architecture for Clusters*, Denver, CO, April 2005.

[4] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev Thakur. PVFS: A parallel file system for Linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, 2000.

[5] James Griffioen and Randy Appleton. Reducing file system latency using a predictive approach. In *USTC'94: USENIX Summer Tech. Conf.*, volume 1, pages 13–13, 1994.

[6] Kamil Iskra, John W. Romein, Kazutomo Yoshii, and Pete Beckman. ZOID: I/O-forwarding infrastructure for petascale architectures. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 153–162, 2008.

[7] Edmund B. Nightingale, Peter M. Chen, and Jason Flinn. Speculative execution in a distributed file system. In *Proceedings of the twentieth ACM Symposium on Operating Systems Principles (SOSP)*, pages 191–205, 2005.

[8] Brian Pawlowski, Chet Juszczak, Peter Staubach, Carl Smith, Diane Lebel, and Dave Hitz. NFS version 3: Design and implementation. In *USENIX Summer Tech. Conf.*, pages 137–152, 1994.

[9] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Seshan. Measurement and analysis of tcp throughput collapse in cluster-based storage systems. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST)*, pages 1–14, 2008.

[10] D. Roselli, J. Lorch, and T. Anderson. A comparison of file system workloads. In *Proc. of the 2000 USENIX Ann. Tech. Conf.*, pages 41–54, June 2000.

[11] R. Srinivasan. XDR: External Data Representation Standard. rfc 1832, IETF, August 1995.