POSTER: An Innovative Storage Stack Addressing Extreme Scale Platforms and Big Data Applications

Jay Lofstead*, Ivo Jimenez†, Carlos Maltzahn†, Quincey Koziol‡, John Bent§, Eric Barton¶
*Sandia National Laboratories gflofst@sandia.gov
†University of California, Santa Cruz ivo@cs.ucsc.edu, carlosm@soe.ucsc.edu
‡HDF Group koziol@hdfgroup.org, §EMC john.bent@emc.com, ¶Intel eric.barton@intel.com

I. Introduction

Current production HPC IO stack design is unlikely to offer sufficient features and performance to adequately serve extreme scale science platform requirements as well as Big Data problems.

A joint effort between the US Department of Energy's Office of Advanced Simulation and Computing and Advanced Scientific Computing Research commissioned a project to develop a design and prototype for an IO stack suitable for the extreme scale environment. It will be referred to as the Fast Forward Storage and IO (FFSIO) project. This is a joint effort led by Lawrence Livermore National Laboratory, with the DOE Data Management Nexus leads Rob Ross and Gary Grider as coordinators and contract lead Mark Gary. The participating labs are LLNL, SNL, LANL, ORNL, PNL, LBNL, and ANL. Additional industrial partners contracted include the Intel Lustre team, EMC, DDN, and the HDF Group. This team has developed a specification set [5] for a future IO stack to address the identified challenges.

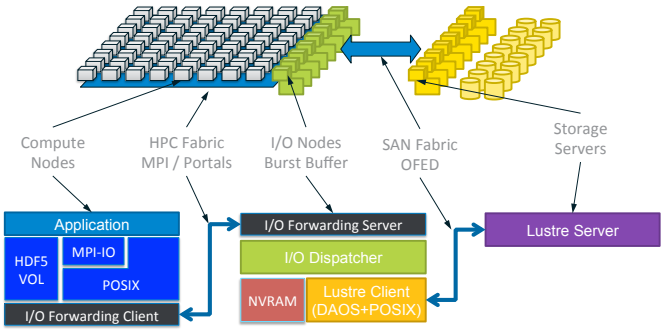

Fig. 1. Target Architecture and Component Mapping

Overall, the architecture shifts from the idea of files and directories to containers of objects. At a more detailed view, the various layers of the IO stack each contribute different functionality. The architecture (Figure 1) incorporates five layers, some of which have potentially optional components. The top layer is generally a high level IO library, such as the demonstration HDF5 library [8]. It is in dark blue. Below the user API is an IO forwarding layer that redirects IO calls from the compute nodes to the IO dispatching layer (in black). This IO forwarding layer is analogous to the function of the IO nodes in a BlueGene machine or the passive data staging processes demonstrated previously [6], [1]. The next two layers have considerable functionality. The IO dispatcher (IOD) serves as the primary storage interface for the IO stack (in green) and offers features like Burst Buffers to insulate the persistent storage array from bursty IO workloads. Ideally, the IOD layer's functionality can be optional based on available hardware and compute power provided on the IO Nodes (IONs). Much of the functionality offered at this layer would shift either up or down the stack as discussed in detail below. The Distributed Asynchronous Object Storage (DAOS) layer serves as the persistent storage interface and translation layer between the user-visible object model and the requirements of the underlying storage infrastructure. It is intended to be the traditional file system-like foundation on which everything else is built with no dependence on any technologies specified above it (in dark pink and yellow). At the bottom is the Versioning Object Storage Device (VOSD) (in purple). It serves as the interface for storing objects of all types efficiently for each storage device in the parallel storage array. Think of this layer as the physical disk interface layer.

II. End-User API Layer

Since the proposal specifies a high-level IO API will be the primary end-user interface for programmatically interacting with the FFSIO stack, the team used the HDF5 API and leveraged the underlying Virtual Object Layer (VOL) to shift from writing to an HDF5 file to using the FFSIO IOD or DAOS interface. This also serves as a good test determining what are strictly necessary extensions to an existing IO API to support the new functionality. The additional functionality, such as transactions, can be ignored for legacy implementations, but these applications will not be able to take advantage of the asynchronous IO support inherent to the new API. The additions comprise API extensions and function analysis shipping from compute nodes to IO nodes.

Since HDF5 has traditionally offered an interface focused on files and the internal data types, such as datasets, these concepts must be mapped onto the proposed FFSIO data storage concepts. At a high level, think of the HDF5 types mapping to FFSIO types as follows: file $\rightarrow$ container, dataset $\rightarrow$ array, and groups and attributes $\rightarrow$ key-value store.

III. IO Forwarding Layer

The IO Forwarding layer offers a mechanism to reduce the concurrency impact of the massive process count on the storage stack. The BlueGene platform incorporated dedicated hardware to perform this role. The proposed functionality for this layer, beyond managing the number of connections to the IOD layer, is to implement function shipping from the compute nodes to the IO nodes.

IV. IO Dispatcher Layer

Strictly speaking, the IO Dispatcher layer and included functionality, such as burst buffers, is optional. All of the functionality can be handled by other portions of the stack.

IOD has three main purposes. First, the burst buffers work as a fast cache absorbing write operations that then trickles out to the central storage array. It can also be used to retrieve objects from the central storage array for more efficient read operations and offers data filtering to make client reads more efficient. Second, it offers the transaction mechanism for controlling data set visibility and to manage faults that could expose an incomplete or corrupt data set to users. These transactions are local to the IOD layer until persisted to the DAOS layer eliminating the need for burdening the persistent storage with transient data. Third, data processing operations can be placed in the IOD. These operations are intended to offer functionality like data rearrangement and filtering prior to data reaching the central storage array.

A. FFSIO Data Model Types

With the shift from a directories and stream-of-bytes files model to the container and object model, some description is required to better understand how these concepts are being used as well as the raw benefits.

A container is similar to a file, but stored within a hash space rather than a hierarchy. It holds a collection of objects that are treated as a unit in isolation.

The base type for the container is a key-value store. A tree of key-value stores are used to represent the hierarchy within an HDF5 file with objects linked into various levels of the tree.

Data is stored as either a blob or a multi-dimensional array. The main difference between the two is that by annotating an object as an array, advanced data management operations can be applied.

B. Burst Buffers

The idea of burst buffers were initially explored in the context of data staging [2], [1], [6], [9]. These initial designs all use extra compute nodes to represent the data storage buffer given the lack of any dedicated hardware support for this functionality. The desired outcome of these initial studies is to motivate how such functionality might be incorporated and the potential benefits. Later, these concepts were incorporated into the existing IO stack architecture [7], [4], [3].

V. DAOS Layer

The Distributed Asynchronous Object Storage layer serves as the traditional parallel file system interface layer for the storage devices. This is the consistent, global view of the underlying devices represented in this stack by the VOSD layer.

This is the layer where the container/object model is translated into the physical storage requirements dictated by the physical storage underneath (the VOSD layer). The two key design elements of this layer are the handling of epochs and the mapping of containers and objects to the underlying storage.

To address consistency issues between platforms, containers at the DAOS layer must know of every transaction.

VI. VOSD Layer

The Versioning Object Storage Device (OSD) layer operates as the interface for each persistent storage device used to support the parallel storage array. In the purest form, it uses a local file system to arrange storage of objects that represent parts of the higher level objects in containers.

VII. Demonstration

This stack has an early prototype implementation intended to test concepts rather than performance and scalability. It has focused on examining the interaction of the different APIs for each layer to flesh out any detailed requirements or concerns that may have been missed in the conceptualization of this IO stack. To demonstrate the viability of the IO stack described in this paper, we show some very early performance results from the untuned prototype.

All of the tests are performed on the Buffy Cray XC30-AC at LANL.

The results (graphs in poster) show the performance of reading and writing different sizes for 56 clients, the smallest client count when performance stabilizes in the number of hosts tests (presented on the poster). The performance of both of these tests are reported to give a very rough idea of the overhead that might be involved. Rather than a true overhead, this should be considered the maximum overhead that should be expected once an optimized, fully functional IO stack is deployed without relying on translating to an underlying parallel file system.

VIII. Acknowledgements

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.

References

[1] H. Abbasi, J. Lofstead, F. Zheng, S. Klasky, K. Schwan, and M. Wolf. Extending i/o through high performance data services. In *Cluster Computing*, Luoisiana, LA, September 2009. IEEE International.

[2] H. Abbasi, M. Wolf, and K. Schwan. LIVE data workspace: A flexible, dynamic and extensible platform for petascale applications. In *CLUSTER '07: Proceedings of the 2007 IEEE International Conference on Cluster Computing*, pages 341–348, Washington, DC, USA, 2007. IEEE Computer Society.

[3] J. Bent, S. Faibish, J. Ahrens, G. Grider, J. Patchett, P. Tzelnic, and J. Woodring. Jitter-free co-processing on a prototype exascale storage stack. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1–5, April 2012.

[4] J. Bent, G. Grider, B. Kettering, A. Manzanares, M. McClelland, A. Torres, and A. Torrez. Storage challenges at los alamos national lab. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1–5, April 2012.

[5] Fastforward storage and i/o stack design documents. Intel FastForward Wiki, February 2014. https://wiki.hpdd.intel.com/display/PUB/Fast+Forward+Storage+and+IO+Program+Documents.

[6] A. Nisar, W.-k. Liao, and A. Choudhary. Scaling parallel I/O performance through I/O delegate and caching system. In *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[7] P. Nowoczynski, N. Stone, J. Yanovich, and J. Sommerfield. Zest checkpoint storage system for large supercomputers. In *Petascale Data Storage Workshop, 2008. PDSW '08. 3rd*, pages 1 –5, nov. 2008.

[8] The HDF Group. Hierarchical data format version 5, 2000-2014. http://www.hdfgroup.org/HDF5.

[9] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. PreDatA - preparatory data analytics on Peta-Scale machines. In *In Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium, April, Atlanta, Georgia*, 2010.