A Plugin for HDF5 using PLFS for Improved I/O Performance and Semantic Analysis

Kshitij Mehta*, John Bent†, Aaron Torres‡, Gary Grider‡, Edgar Gabriel*
* Department of Computer Science at University of Houston † EMC Corporation ‡ Los Alamos National Laboratory

Abstract—HDF5 is a data model, library and file format for storing and managing data. It is designed for flexible and efficient I/O for high volume and complex data. Natively, it uses a single-file format where multiple HDF5 objects are stored in a single file. In a parallel HDF5 application, multiple processes access a single file, thereby resulting in a performance bottleneck in I/O. Additionally, a single-file format does not allow semantic post processing on individual objects outside the scope of the HDF5 application. We have developed a new plugin for HDF5 using its Virtual Object Layer that serves two purposes: 1) it uses PLFS to convert the single-file layout into a data layout that is optimized for the underlying file system, and 2) it stores data in a unique way that enables semantic post-processing on data. We measure the performance of the plugin and discuss work leveraging the new semantic post-processing functionality enabled. We further discuss the applicability of this approach for exascale burst buffer storage systems.

Index Terms—HDF5, PLFS, semantic analysis

1 Introduction

Hierarchical Data Format (HDF5) is a technology suite for efficient management of large and complex data [7]. It supports complex relationships between data and dependencies between objects. HDF5 is widely used in industry and scientific domains, in understanding global climate change, special effects in film production, DNA analysis, weather prediction, financial data management etc. [10] Parallel HDF5 (PHDF5) [3] enables developing high performance, parallel applications using standard technologies like MPI in conjunction with HDF5. PHDF5 exports a standard parallel I/O interface which itself uses MPI's parallel I/O functionality. This is used along with parallel file systems to achieve high performance I/O.

An HDF5 file is a self-describing format which combines data and metadata. It is a container in which users typically store multiple HDF5 objects alongside their metadata. A PHDF5 application has multiple processes accessing a single file (i.e. N-1 access pattern). Unfortunately, many popular parallel file systems are known to behave poorly under these circumstances [2, 12]. Secondly, HDF5 stores multiple data objects in a single file. This complicates performing useful semantic operations on individual objects.

PLFS (Parallel Log-Structured File System) is a middleware virtual file system developed at Los Alamos National Lab (LANL) [5]. It converts writes to a shared logical file into writes to multiple physical files to overcome the performance bottleneck associated with N-1 writes. However, it suffers from an inability to understand the structure of the data that it stores.

In this paper, we enhance the performance of HDF5 by addressing the performance issues inherently arising from poor file system performance on N-1 access patterns, and augment performing useful post-processing on HDF5 objects. We have developed a new plugin for HDF5 using its recently introduced Virtual Object Layer (VOL). This plugin stores data in a unique way that enables semantic post-processing on HDF5 objects, and uses PLFS to convert N-1 accesses into N-N accesses, thereby resulting in improved I/O performance.

2 Background

HDF5 is a versatile data model containing complex data objects and metadata. Its information set is a collection of datasets, groups, datatypes and metadata objects. The data model defines mechanisms for creating associations between various information items. The main components of HDF5 are described below.

File: In the HDF5 data model the container of an HDF5 infoset is represented by a file. It is a collection of objects that also explains the relationship between them. Every file begins with a root group "/", which serves as the "starting-point" in the object hierarchy.

Dataset: HDF5 datasets are objects that represent actual data or content. Datasets are arrays which can have multiple dimensions. A dataset is characterized by a dataspace and a datatype. The dataspace captures the rank (number of dimensions), and the current and maximum extent in each dimension. The datatype describes the type of its data elements.

Group: A group is an explicit association between HDF5 objects. It is synonymous with directories in a file system. A group could contain multiple other groups, datasets or datatypes *within* it.

Attribute: Attributes are used for annotating datasets, groups, and datatype objects. They are datasets themselves, and are *attached* to existing objects they annotate.

For example, as shown in figure 1, the file "Sample.h5" contains the root group which itself contains a group G1 and two datasets, D1 and D2. Group G1 contains a dataset D3. Attribute A1 is linked to dataset D1. The objects and the relationships between them can be represented as a B-tree, which is used internally by HDF5 to index its objects.

Virtual Object Layer (VOL): VOL is a new abstraction layer internal to the HDF5 library [4]. As shown in figure 2 it is implemented just below the public API. The VOL exports an interface that allows writing plugins for HDF5, thereby enabling developers to store objects in a format different from the default HDF5 file format (like native netCDF or HDF4 format). Plugin writers provide an implementation for a set of functions that access data on disk. These include functions for file management, dataset creation and access, group creation, to name a few.

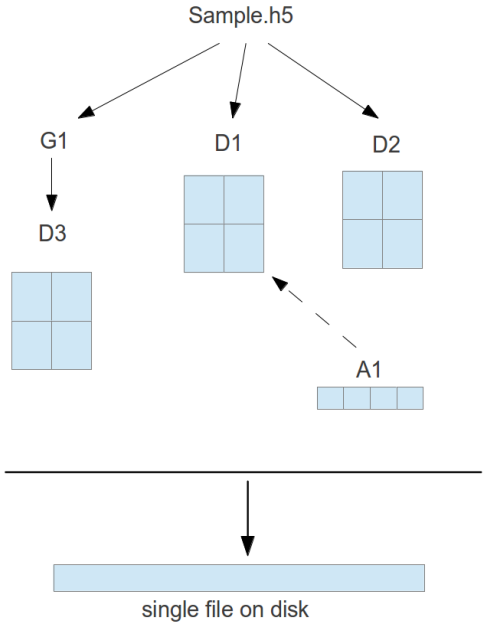

Fig. 1. A sample HDF5 file

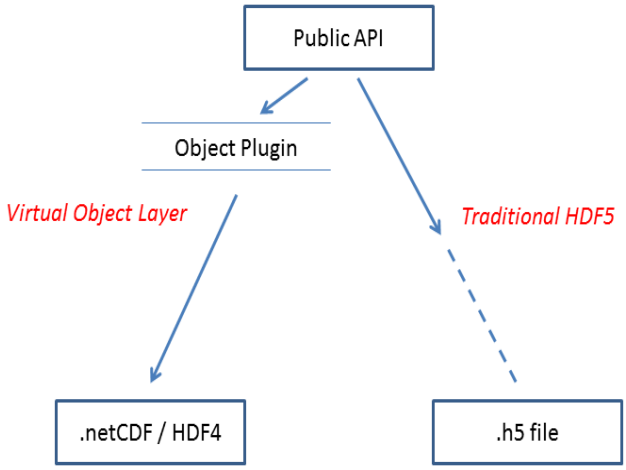

Fig. 2. Virtual Object Layer (VOL)

PLFS is a middleware virtual file system that converts writes to a shared logical file into writes to multiple physical files. It is situated between the application and the parallel file system responsible for the actual data storage. It transforms N-1 into N-N, where every process participating in I/O writes data to its own, separate file. The basic operation of PLFS is as follows. For every writer to a logical file, PLFS creates a unique physical file on the underlying

parallel file system. It also maintains sufficient metadata to recreate the shared logical file. We added a new feature to PLFS called extendible attributes (Xattrs). Xattrs serve as short, extensible metadata stored as key-value pairs. They can be used to store user-defined information about data for easy and fast retrieval.

Users can interface with PLFS directly by using the PLFS API or by using its MPI-IO driver (ad_plfs).

As stated before, when multiple processes access a single, shared file, file systems like Lustre are known to exhibit sub-standard I/O performance. PLFS has demonstrated benefits of converting this N-1 access pattern into one where every process writes data to its own physical file and sufficient metadata is maintained to recreate the original file in the correct order of writes issued. Secondly, the data model of HDF5 stores multiple datasets in a single file on disk. As a result, one cannot perform any useful post-processing on individual HDF5 objects. As we shall see, storing objects in a different file format allows us to perform optimizations on individual HDF5 objects without affecting the remaining data in the file.

3 Plugin

We have developed a plugin which stores data in a format which enables performing semantic analysis on the data and uses PLFS for high I/O throughput. Instead of storing all objects in a single file, it stores every HDF5 object in a separate location, so that data from different objects is not contained in the same file. In short, we provide a raw mapping of HDF5 objects to the file system. HDF5 files and groups are stored as directories, whereas datasets and attributes, which contain raw data, are stored in files created using PLFS. Consider the example shown in figure 1. Using the plugin, file Sample.h5 is stored as a directory. Group G1 is stored as a directory under it, and datasets D1 and D2 are files. Attribute A1 is a file created at the same location as dataset D1 to which it is attached. Additionally, the name of the attribute is stored as *dataset name.attribute name* to denote

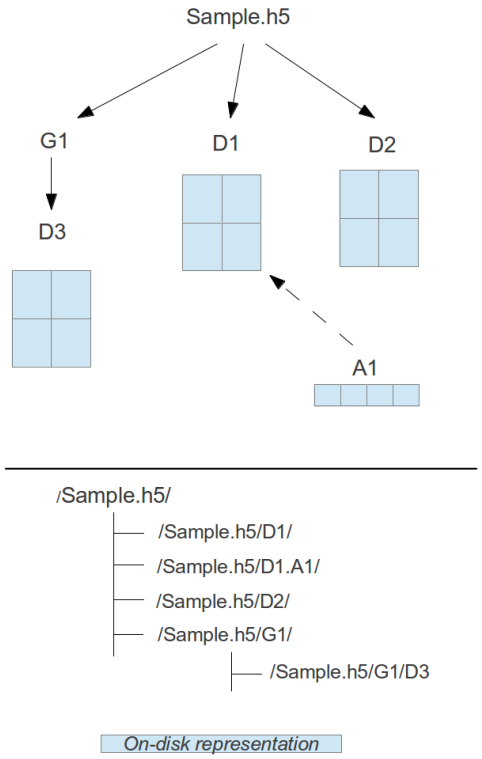

Fig. 3. HDF5 data stored using the plugin

the dataset to which the attribute is attached. Thus these objects are stored at the following paths as shown in figure 3:
/Sample.h5/
/Sample.h5/G1
/Sample.h5/G1/D3
/Sample.h5/D1
/Sample.h5/D1.A1
/Sample.h5/D2

We can see that the relationship between the objects is represented by their relative paths at the file system. That is, the path /Sample.h5/G1/D3 tells us that D3 is a dataset belonging to Group G1 under the root group of the file. This approach gives us the ability to distinguish HDF5 objects inside the storage system. Also, the plugin eliminates the need to explicitly store the metadata describing the relationship between objects. Metadata about datasets, such as the datatype, extent, dimensions etc. are stored as Xattrs.

Our plugin makes direct calls to the PLFS API. For the current version, we do not support collective I/O operations. It should be noted that HDF5 when used natively with MPI-IO

allows users to specify whether collective I/O should be used for reading and writing data.

The above format of storing data allows us to perform at least two different analysis optimizations. To illustrate via an example, imagine storing a three-dimensional ocean model within a PLFS file. The storage system sees the file as an opaque linear array of bytes. With the structure, however, PLFS can provide *active analysis* as well as *semantic restructuring*.

Active analysis borrows the transducers idea from the Semantic File System [9] which has since been productized in Google's BigTable and Apache Hbase technologies [6, 8, 14]. With active analysis, the application can ship a data parser function when it creates the PLFS file. As the data is written into PLFS, PLFS can apply the data function on the streaming data. The function will output key-value pairs which PLFS can embed in its extensible metadata (figure 4). In this example, one simple function might record the height of the largest wave. Due to PLFS's model of storing a logical file across multiple physical files, the PLFS extensible metadata can record the height of the largest wave within each physical file. However, given that PLFS now understands the structure of the logical file, these multiple physical files, within the PLFS container, are more accurately thought of as shards. In a future burst buffer architecture [11], these semantic shards will be spread across multiple burst buffer nodes. Therefore, subsequent analysis of the ocean model can quickly find the burst buffer containing the shard with the largest wave by searching a small amount of extensible metadata instead of scanning the entire ocean model.

Semantic restructuring is the idea of reorganizing the data into a new set of semantic shards. This would be done to speed future analysis routines. For example, assume that the ocean model was originally sharded using a row-order organization (i.e. across latitude instead of longitude). An analysis routine which will explore the model along a column-ordering will suffer poor performance with the row-order organization as its access pattern will result in a large number of small reads from a large set of semantic shards. However,

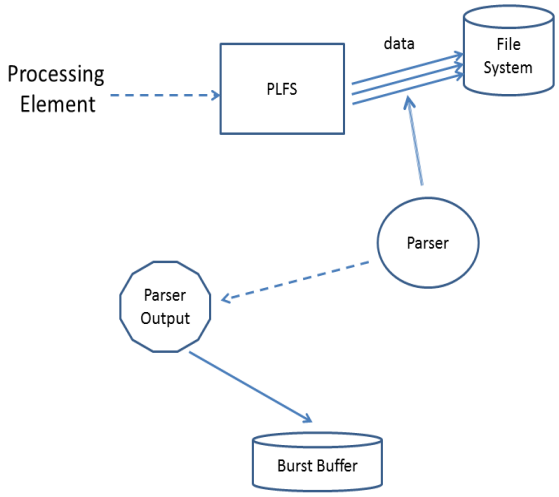

Fig. 4. Active Analysis of Data

by knowing the semantic structure, the analysis routine can request a semantic restructuring (figure 5) which will be a compact, intuitively described request such as "restructure into column-ordering." Without structural knowledge, a semantic restructuring would be significantly more complicated: the analysis routine would have to send a large list of logical offsets to PLFS to inform it of expected read patterns. In an exascale system, the list of logical offsets will be in the order of one billion. Semantic restructuring shrinks the size of the request to a small constant value. A storage format where datasets are stored in separate files allows us to make modifications to the storage layout of one dataset without affecting other datasets and data.

4 Evaluation

In this section, we present the the I/O performance of our plugin. These experiments are not targeted towards performing a semantic analysis of data, such as active analysis or semantic restructuring discussed in the previous section. For evaluation purposes, we have used HDF5's *h5perf* performance tool [1]. It allows configuring various parameters, such as the number of processes, number of datasets, amount of data read/written by a process in a single I/O call (transfer size) etc. For our measurements,

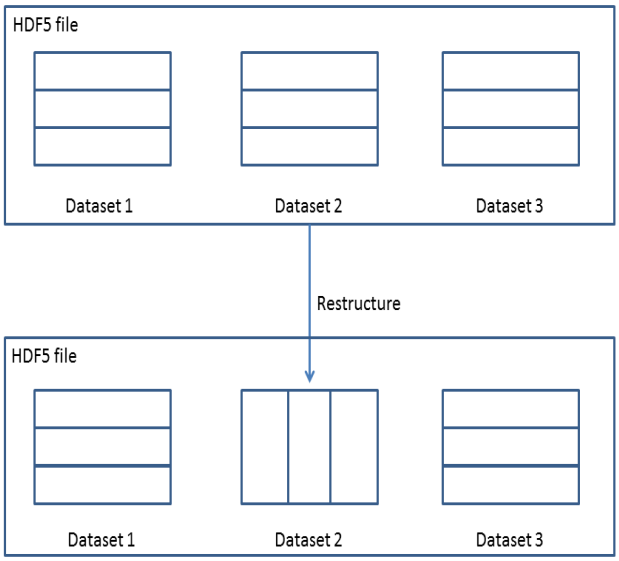

Fig. 5. Semantic Restructuring of Data

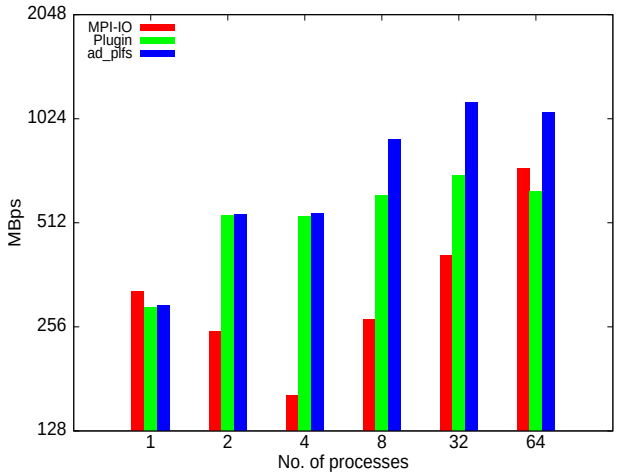

Fig. 6. Performance of contiguous writes

we created 10 1-dimensional datasets, and the total file size was 64 GB or more. In every run, every process contributed equal amount of data per dataset using the default individual (non-collective) I/O mode of h5perf.

Tests were performed on the Lustre parallel file system [13] on the Atlas cluster at University of Dresden. The file system has 12 OSTs with a stripe size of 1MB. The file system is connected to the compute nodes using an SDR Infiniband link. The cluster has 92 AMD Opteron nodes with 64 cores each and 64 to 512 GB memory. Tests were run thrice and we present the average bandwidth values, which does not include the time taken to open and close the file.

We have performed tests with 1,2,4,8,32 and 64 processes with a maximum of 4 processes per node. Reads and writes are either contiguous or interleaved; processes either access contiguous locations in file or execute a strided pattern. We compare the performance of the default MPI-IO driver, our plugin, and the PLFS MPI-IO driver (ad_plfs).

In Figure 6, we show the write performance for a transfer size of 1MB for contiguous writes. It can be seen that the plugin regularly outperforms MPI-IO except for the 64 process case, where the metadata overhead incurred by the plugin is high. The performance of ad_plfs is the best for higher process counts. Figure 7 shows the performance of contiguous reads. Figures 8 and 9 show the interleaved write and read performance respectively for an unaligned transfer size (1M + 10bytes) for a maximum of 8 processes. The write performance of MPI-IO is quite poor in this case. The plugin easily outperforms MPI-IO and almost matches the performance of ad_plfs.

Overall, results show that the plugin consistently shows good performance, however it does not scale as well as ad_plfs. This is due to the fact that since we are directly making calls to the PLFS API, the metadata overhead is high. All processes participate in file open and close operations, which for large problem sizes adds significant overhead. Hence, for future work, we plan to use ad_plfs directly in the plugin which can overcome this performance drawback, since ad_plfs has collective optimizations in open and close. It should be noted that there are some MPI libraries tailored to suit specific types of applications and which do not provide an implementation for MPI-IO. Such libraries can benefit from using a plugin that does not rely on MPI-IO.

5 Conclusion

Using HDF5's virtual object layer, we have developed a new plugin that makes use of PLFS for improved I/O performance. We deviate

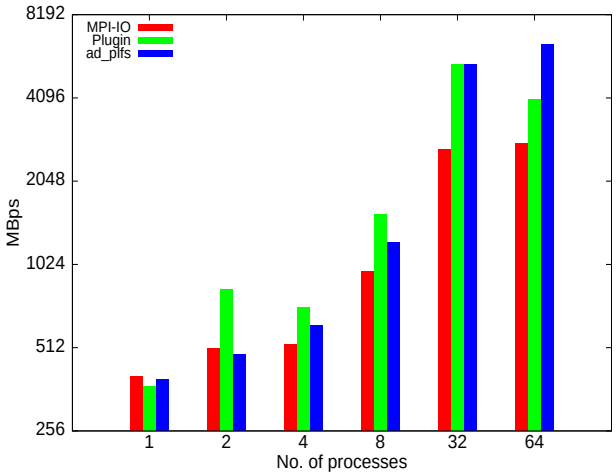

Fig. 7. Performance of contiguous reads

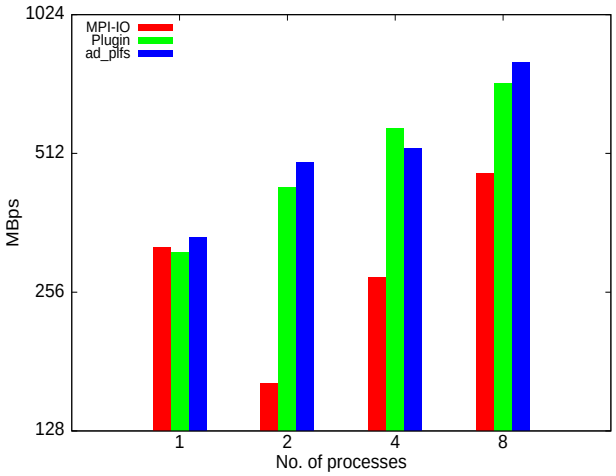

Fig. 9. Performance of interleaved,unaligned reads

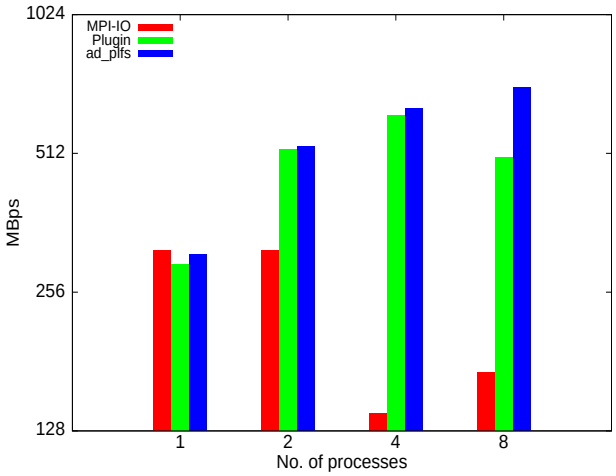

Fig. 8. Performance of interleaved, unaligned writes

from HDF5's native file format and store data in a way that allows us to perform useful post-processing on individual HDF5 objects. Such a format can further allow us to incorporate semantic analysis of data for burst buffer type workloads. Initial results show that our plugin outperforms MPI-IO in most cases, whereas MPI-IO when used with PLFS's MPI-IO driver shows the best results.

ACKNOWLEDGMENTS

We would like to thank Mohamad Chaarawi and Quincey Koziol from the HDF group for their help with the development of the plugin. We would also like to thank University of Dresden, Germany for the use of their Atlas cluster hosting the Lustre file system.

REFERENCES

[1] h5perf User Guide. http://www.hdfgroup.org/HDF5/doc/UG/index.html.

[2] I/O Patterns from NERSC Applications. https://outreach.scidac.gov/hdf/NERSC_User_IOcases.pdf.

[3] Parallel HDF5. http://www.hdfgroup.org/HDF5/PHDF5/.

[4] Virtual Object Layer. https://confluence.hdfgroup.uiuc.edu/display/VOL/Virtual+Object+Layer.

[5] John Bent, Garth Gibson, Gary Grider, Ben McClelland, Paul Nowoczynski, James Nunez, Milo Polte, and Meghan Wingate. Plfs: a checkpoint filesystem for parallel applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 21:1–21:12, New York, NY, USA, 2009. ACM.

[6] Jeff Dean. Designs, Lessons and Advice from Building Large Distributed Systems, 2009. http://www.odbms.org/download/dean-keynote-ladis2009.pdf.

[7] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. An overview of the hdf5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, AD '11, pages 36–47, New York, NY, USA, 2011. ACM.

[8] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[9] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole, Jr. Semantic file systems. In *Proceedings of the thirteenth ACM symposium on Operating systems principles*, SOSP '91, pages 16–25, New York, NY, USA, 1991. ACM.

[10] HDF group. HDF5 Users. http://www.hdfgroup.org/HDF5/users5.html.

[11] Ning Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn. On the role of burst buffers in leadership-class storage systems. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1 –11, april 2012.

[12] J. Logan and P. Dickens. Towards an understanding of the performance of mpi-io in lustre file systems. In *Cluster Computing, 2008 IEEE International Conference on*, pages 330 –335, 29 2008-oct. 1 2008.

[13] Lustre webpage. http://www.lustre.org.

[14] Andrew Purtell Mingjie Lai, Eugene Koontz. Apache HBase. https://blogs.apache.org/hbase/entry/coprocessor_introduction.