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

Kshitij Mehta, John Bent, Aaron Torres and Edgar Gabriel

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. The HDF group has recently introduced a new layer called the Virtual Object Layer (VOL) that allows writing plugins for HDF5 to store data in different ways. We have written a new plugin for HDF5 using PLFS that provides 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 the data, independently of HDF5. In this paper, we measure the performance improvements available through the PLFS VOL and discuss future 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 data model , library and file format for storing and managing data. It is designed for flexible and efficient I/O. HDF5 defines an information set, which is a container of array variables, groups, and types. The data model defines mechanisms for creating associations between various information items. HDF5 is widely used in the industry and scientic domain, in understanding global climate change, special effects in film production, DNA analysis, weather prediction, financial data management etc. (citation here) Parallel HDF5 (PHDF5) enables developing high performance, parallel applications using standard technologies like MPI in conjunction with HDF5. An HDF5 file created using PHDF5 is compatible with serial HDF5 files and is shareable between different platforms. PHDF5 exports a standard parallel I/O interface which then uses MPI's parallel I/O functionality.

The central component of HDF5, the file, is a self-describing format which combines data and metadata. Users typically store multiple HDF5 objects in a single file, and the library stores metadata alongwith data that describes the relationships between various objects amongst other metadata. However, this native single-file format has its disadvantages. A PHDF5 application has multiple processes accessing a single HDF5 file. Many popular parallel file systems are known to behave poorly under these circumstances. Secondly, since many HDF5 objects are stored in a single file, this eliminates any possibility of performing useful semantic analysis on objects beyond the scope of an HDF5 application. In this paper, we address these shortcomings of HDF5, viz. performance issues due to multiple processes accessing a single shared file, and lack of a way to perform useful post-processing on HDF5 objects inherent due to the native file format.

PLFS is a middleware virtual file system developed at Los Alamos National Lab (LANL). It converts writes to a shared logical file into writes to multiple physical files. Thus, it interposes on application I/O and converts its I/O pattern into one that is more suitable for the underlying parallel file system. PLFS is popularly used as a checkpoint file system where applications demonstrating heavy checkpoint workloads are known to benefit significantly from PLFS.

In this work, we have developed a new plugin for HDF5 using its recently introduced Virtual Object Layer (VOL). The VOL exports an interface that allows writing plugins for HDF5, where it primarily passes references to HDF5 objects, thereby enabling developers to store objects in a format different from the default HDF5 file format. We have written a plugin with two main objectives. The plugin stores data in a unique way that enables semantic post-processing on HDF5 objects outside the scope of the HDF5 application. Secondly, PLFS converts N–¿1 accesses into N–¿N accesses, thereby showing a significant improvement in performance as well. Preliminary results using HDF5's h5perf performance tool show a Xx performance improvement in I/O overall traditional PHDF5.

The rest of the paper is organized as follows. We represent details about HDF5 and PLFS in section 2, describe our plugin design and implementation in section 3, and our evaluation in section 4. We present related work in section 5, the current status of the plugin alongwith future work in section 6, and finally we conclude in section 7.

2 Background

2.1 HDF5

HDF5 is a data model , library and file format for storing and managing data. It defines an information set, which is a container of array variables, groups, and types. They are known as datasets, groups and datatype objects in HDF5 respectively. The data model defines mechanisms for creating associations between various information items. The main components of HDF5 are briefly described below.

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

Datasets: 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. An HDF5 dataspace captures the rank (number of dimensions) , and the current and maximum extent in the respective dimensions. An HDF5 datatype describes the type of its data elements. HDF5 supports ten classes of datatypes including scalar types like integer, floating-point, string, etc. and compound datatypes.

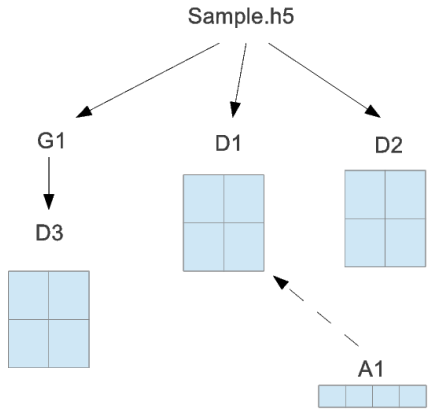

Fig. 1. A sample HDF5 file

Groups: HDF5 groups are akin to directories in a file system (reference here). A group is an explicit association between HDF5 objects. A group could contain multiple other groups, datasets or datatypes /textitwithin it. Traversing a root group generates the /information set graph/.

Attributes: Aattributes are used for annotating datasets, groups, and datatype objects. They are datasets themselves, and are /textitattached to the existing objects they annotate.

The above definitions are explained in the example shown in figure 1. The file "sample.h5" contains a 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 datatype in all datasets in an integer.

HDF5 file format: An HDF5 file is a self-describing format which combines data and metadata. High level objects are stored along-

with metadata that is used to describe various characteristics about the object, alongwith the relationship between various objects.

Parallel HDF5 (PHDF5): As described previously, HDF5 supports parallelism using MPI. Multiple processes can be used to create and store objects in a file. PHDF5 exports a parallel I/O interface, where internally makes use of MPI-IO to perform I/O in parallel. Applications typically store multiple objects in a single HDF5 file. However, many popularly used parallel file systems like Lustre, Panasas etc. are known to perform poorly for workloads where multiple processes access a shared file (commonly referred to as the N–¿1 access pattern).

2.2 PLFS

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 an application's N–¿1 access pattern into an N–¿N access pattern, where every process participating in I/O writes data to its own , separate file. Thus, it interposes on application I/O and converts its I/O pattern into one that is more suitable for the underlying parallel file system. The basic operation of PLFs is as follows. For every logical file, PLFS creates a /texitcontainer structure on the underlying parallel file system. This container is a hierarchical directory tree consisting of multiple sub-directories. Multiple processes opening the same logical file for writing get a unique data file within the container. When the processes write to their file, a record identifying the write is appended to an index file shared between all processes. Thus, PLFS maintains sufficient metadata to recreate the shared logical file. A FUSE daemon exports this container structure as a single, logical file to the end-users, thereby keeping the PLFS operations transparent from them.

3 PLUGIN

The plugin stores data in a format which enables performing semantic analysis on the data. It stores every HDF5 object in a separate file/directory, so that data from different objects is not contained in the same file. The central object store in HDF5, the HDF5 file, is represented by a directory. Groups are stored as directories as well, whereas datasets, which contain raw data, are stored in files. The relationship between objects is explained by the relative paths between the objects at the file system level. As an example, the HDF5 object D1 which in HDF5's B-tree format can be represented as /G1/G2/D1 is stored on the file system as /path-to-hdf5-file/G1/G2/D1. This way , the plugin eliminates the need to store the metadata describing the relationship between objects. Metadata about datasets, such as the datatype, extent, dimensions etc., are stored as PLFS Xattributes (xattrs) as explained in the previous section.

Consider the example shown figure 1. Using the plugin, file test.h5 is stored as a directory. Groups G1, G2, G3 are represented as directories at the paths /tmp/test.h5/G1, /tmp/test.h5/G2 , /tmp/test.h5/G1/G3/ respectively. Datasets D1 and D2 are files stored under their respective groups.

This approach gives us the ability to distinguish HDF5 objects at the file system level, without requiring an HDF5 application to provide information about objects and the relationship between them. We can now perform post-processing and semantic analysis on the data, outside the scope of the HDF5 application.