HDF5 VOL Connector to Apache Arrow

Jie Ye
Illinois Institute of Technology
Chicago, USA
jye20@hawk.iit.edu

Anthony Kougkas
Illinois Institute of Technology
Chicago, USA
akougkas@iit.edu

Xian-He Sun
Illinois Institute of Technology
Chicago, USA
sun@iit.edu

I. Extended Abstract

With the ever-increasing dataset sizes and volumes, various kinds of file formats such as Apache Parquet [1], ORC [2], Avro [3] and Apache Arrow [4], [5] have been developed to store data efficiently. In recent years, Apache Arrow is very popular in Big Data Analysis and Cloud Computing domain due to the standard columnar in-memory data representation and its efficient data processing and data transfer. The columnar data layout enables to take advantage of the SIMD (Single Instruction, Multiple Operations) operation in modern computers. It also reduces overhead of the copy-and-convert when moving the data from one system to another. As Apache Arrow is considered efficient and the data analysis can be accelerated with integration of it. Thus, there is a need to verify if the Apache Arrow can be used in High Performance Computing system.

However, most scientific applications currently prefer to using HDF5 [6], a widely used I/O middle-ware on HPC sytems, to store and manage data. HDF5 is designed to store and manage high-volume and complex scientific data. Although it supports a variety of features, like random access to individual objects, partial access to selected dataset regions and internal data compression, it doesn't support column storage and inefficient for column-access.

As mentioned above, Apache Arrow can create an efficient in-memory column store that can be used to manage streamed data. Accessing this data through HDF5 calls would applications to take advantage of transient, column-oriented data streams, such as real-time data through high-speed instruments and cameras. Moreover, bridging the gap between science applications and analytic tools that use HDF5 and Apache Arrow could bring new kinds of kinds of data together. Therefore, there is a need to create a tool to support accessing Apache Arrow data through native HDF5 calls without changing the applications. Thus, the objects of this work are:

1) Design and implement a HDF5 VOL connector which allows applications to access Apache Arrow data through native HDF5 calls.
2) Explore its use for analyzing scientific data

II. Background

A. HDF5

HDF5 is a well-established and very flexible data model, parallel I/O library and file format, which could handle many store options and needs. HDF5 is designed to store and manage high-volume and complex scientific data. It provides a rich set of pre-defined datatypes as well as an unlimited variety of complex user-defined datatypes. It also supports a lot of powerful features for managing data, such as random access to individual objects, partial access to selected dataset regions and internal data compression. Due to its portability, efficiency and the flexibleness of its data model, HDF5 has been widely-used by a great number of scientific and industry applications in HPC community to store and manage the data they produced. By default, HDF5 library use its native file format when storing data and takes advantage of MPI-IO to perform parallel I/O.

With the emergence of new file format and storage system which is not comply with the POSIX I/O standard, there is a need to support the new file format and storage system without modifying the applications' code. To provide this capability and allow developers to store data with more choices, HDF5 library introduces a Virtual Object Layer (VOL) released in 1.12 version of the library. VOL is a storage abstraction layer within the HDF5 library and is implemented just below the public HDF5 API. The VOL enables applications to store the HDF5 data in many different storage (like storing the data in PDC, DAO, Hermes, etc.) by intercepting the HDF5 I/O API calls and then seamlessly re-routing them to the corresponding VOL connector backend, which could translate these calls into the operations that it desires to perform.

B. Apache Arrow

Apache Arrow is an open source, columnar, in-memory data representation that enables analytical systems and data sources to exchange and process data in real-time. It specifies a standard columnar in-memory format foe representing the structured, table-like datasets. In most cases, Apache Arrow acts as an interface between different computer programming languages and systems. With the columnar in-memory data layout for memory processing, it could process large amounts of data quickly by using SIMD (Single Instruction, Multiple Data) operations. Moreover, due to the standard column format, it reduces the unnecessary overhead copy-and-covert cost when moving data from one system to another. Apache Arrow has a rich set of data types, including nested and user-defined data types. It also creates a in-memory Plasma Object Store [7] for different applications to share data within the same node and makes use of Arrow Flight [8], an RPC

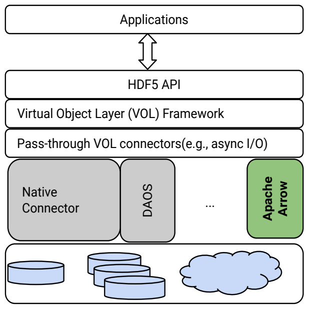

Applications
HDF5 API
Virtual Object Layer (VOL) Framework
Pass-through VOL connectors(e.g., async I/O)
Native Connector
DAOS
...
Apache Arrow

Fig. 1. Apache Arrow within VOL

framework, for high-performance data services based on Arrow data. All of this benefits and advantages make Apache Arrow very popular in Big Data Analysis Area.

III. Arrow VOL Connector Design and Implementation

According to the introduction of Apache Arrow, we already know that Apache Arrow could create an in-memory store and is considered as efficient to store and manage streamed data. Accessing this data through HDF5 API would allow applications to take advantage of the transient, column-oriented data streams, such as real-time data from high-speed scientific instruments and cameras. Therefore, there is a need to support accessing Arrow data through HDF5 calls. Figure 1 shows the Apache Arrow location within VOL. It is a terminal VOL connector, locating at the last layer of all the VOLs. Arrow VOL connector will intercept the related HDF5 I/O API calls and then translates them into Apache Arrow API and saves the data as Apache Arrow tables.

Currently, our Arrow VOL connector only implements a subset of the HDF5 API. Apache Arrow itself is not an engine or storage, it is only a columnar data representation format. Therefore, in our implementation, HDF5 files and groups are mapped to directories and responding sub-directories, while HDF5 datasets are mapped to Apache Arrow tables. In addition, as Apache Arrow doesn't support multiple processes to write part of a single table, each MPI process will write its sepcified subset region as an arrow table to the back-end storage (including parallel file system and Apache Arrow Plasma in-memory Object Store) or Arrow Flight Server. Figure 2 presents the internal work-flow of write and read operations in Arrow VOL connector.

1) **Write Operation**: Once Arrow VOL connector intercepts a *H5Dwrite()* request, it creates an internal column-major buffer and then fill the row-major input data into the internal buffer. This step requires the row-major to column-major conversion. Both the metadata and the internal column-major buffer are kept in memory.
2) **Close Operation**: When Arrow VOL connector intercepts a *H5Dclose()* request, it will create a corresponding Arrow table by the internal buffer and then flush the metadata and the Arrow table into the selected back-end storage.

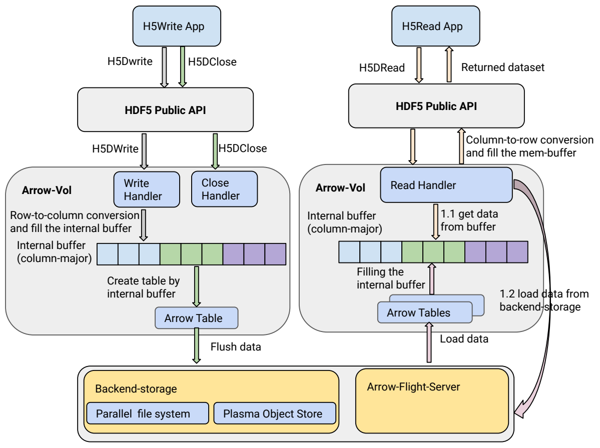


Fig. 2. Internal workflow in Arrow VOL Connector

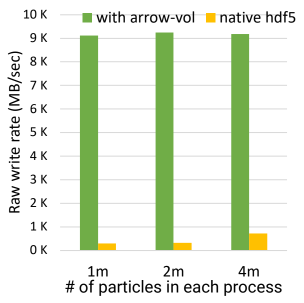
3) **Read Operation**: When intercepting a *H5Dread()* request, Arrow VOL connector will first check if the data is exist in the internal column-major buffer. If the request data has already existed, Arrow VOL connector will fill the output buffer directly through the internal column-major buffer. If request data isn't in the internal column-major buffer, it will first load the data from the back-end storage to the Arrow table and fill into the internal buffer. Then it will fill the output buffer by this internal column-major buffer. As with the write operation, this step needs the column-major to row-major conversion no matter the request data exists in the internal buffer or not.

IV. Initial Results

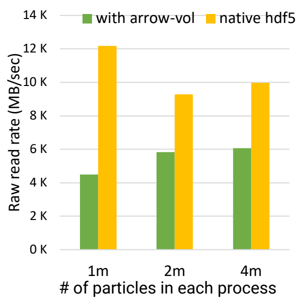
Testbed: All tests were conducted on the Cori Supercomputer at the National Energy Research Scientific Computing Center (NERSC), which is a Cray XC40 supercomputer with 1630 Intel Xeon Haswell nodes. Each node consists of 32 CPU cores and 128GB memory. The supporting storage system is Lustre, an extensively used parallel file system. It has 248 object storage targets (OSTs) and is shared by all users.

Software used: The Arrow VOL connector implementation depends on HDF5 (v1.13.0), Apache Arrow (v4.0.1). The MPICH version used for parallel I/O processing is 3.3.1 and the GCC version is 7.3.0.

Analysis: In our experiment, we evaluated the write performance by VPIC-IO, a plasma-physics application's I/O kernel, and read performance through BD-CAST I/O kernel, which is used for analyzing the data produced by particle simulation. All the experiments were executed on 4 nodes with 128 processes.In VPIC-IO, each MPI process writes a region with different number of particles (such as 1M, 2M and 4M) and each particle has 8 properties. The particles are organized as a 1D-array. Figure 3(a) shows the write performance with arrow-vol and without arrow-vol. We can see that the raw write rate with arrow-vol is about 9 GB/sec while the raw write rate with native hdf5 is only hundreds of Megabytes per second. One reason is because of the different configuration of the file system stripe count for native hdf5 and arrow-vol. The other



(a) Write Performance



(b) Read Performance

Fig. 3. Arrow-VOL vs Native HDF5 performance

reason is that native hdf5 needs to flush data into the file system when executing the *H5write()* operations while the data is saved in memory when using Arrow Vol connector. Figure 3(b) shows the read performance with arrow-vol and without hdf5. We can see that the performance of without arrow-vol far exceeds that of with arrow-vol. That's probably because the test scale is small and part of the data has been cached in memory.

V. Conclusions

In this work, we designed and implemented a HDF5 VOL connector to Apache Arrow that enables science applications to access Apache Arrow data through native HDF5 calls without changing the original code. We also have seen the initial write/read performance results when using Arrow-VOL Connector and native HDF5. Although the performance is not very good, there is still a lot of room for optimization. Most importantly, we have verified that Apache Arrow can be integrated into HPC system, which laid the foundation for our future work, like the integration of HPC and Big Data Analysis.

References

[1] "Apache parquet." [Online]. Available: https://parquet.apache.org/
[2] "Apache orc." [Online]. Available: https://orc.apache.org/
[3] "Apache avro." [Online]. Available: https://avro.apache.org/docs/current/
[4] "Apache arrow." [Online]. Available: https://arrow.apache.org/
[5] J. Chakraborty, I. Jimenez, S. A. Rodriguez, A. Uta, J. LeFevre, and C. Maltzahn, "Towards an arrow-native storage system," *arXiv preprint arXiv:2105.09894*, 2021.
[6] "Hdf5 library." [Online]. Available: https://portal.hdfgroup.org/display/HDF5/HDF5
[7] "Apache arrow plasma in-memory object store." [Online]. Available: https://arrow.apache.org/docs/python/plasma.html
[8] "Apache arrow flight rpc framework." [Online]. Available: https://arrow.apache.org/docs/format/Flight.html