Comparison of Array Management Library Performance - A Neuroscience Use Case

Donghe Kang*, Oliver Rübel†, Suren Byna†, Spyros Blanas*
*Ohio State University, †Lawrence Berkeley National Laboratory
{kang.1002,blanas.2}@osu.edu,{oruebel,sbyna}@lbl.gov

ABSTRACT

Array management libraries, such as HDF5, Zarr, etc., depend on a complex software stack that consists of parallel I/O middleware (MPI-IO), POSIX-IO, and file systems. Components in the stack are interdependent, such that effort in tuning the parameters in these software libraries for optimal performance is non-trivial. On the other hand, it is challenging to choose an array management library based on the array configuration and access patterns. In this poster, we investigate the performance aspect of two array management libraries, i.e., HDF5 and Zarr, in the context of a neuroscience use case. We highlight the performance variability of HDF5 and Zarr in our preliminary results and discuss potential optimization strategies.

1 INTRODUCTION

Scientific data is often organized as multi-dimensional arrays that are stored as files in storage devices. Towards managing I/O and mapping arrays to files, several self-describing data and file format libraries are available. For instance, HDF5 and netCDF have been available for more than two decades and new libraries, such as Zarr are coming to prominence. Evaluating the performance characteristics of these libraries is helpful in selecting a file format.

Since these libraries depend on multiple I/O libraries (POSIX-IO, MPI-IO, and file systems) and have different storage layouts, tuning them for achieving optimal performance is a non-trivial effort. For instance, HDF5 stores everything, including array data and metadata, in a file. In contrast, Zarr stores chunks, fixed-size partitions of an array, as separate files. Because of these fundamental differences of organizing data, their I/O performance varies depending on the application I/O patterns. The storage stack has various parameters impacting the performance, such as chunk shape, stripe count and size, and concurrency.

Several efforts have focused on selecting the I/O optimization parameters file system and MPI-IO, however, selection of array management parameters received less attention. Behzad et al. [1] automatically tunes parameters in the stack to optimize I/O performance of HDF5 applications. However, it does not explicitly explain how these parameters impact the HDF5 performance and is limited to HDF5. Yu et al. [3] characterizes the I/O performance on Lustre file system, which does not cover the full storage stack.

In this poster, we study performance variability of HDF5 and Zarr formats using a neurosicence use case. We use the NWB:N [2] data standard, which supports archiving, processing, and sharing neurophysiology data. We first summarize the common array access patterns in neuroscience area and provide a benchmark to systematically evaluate HDF5 and Zarr and analyze the impact of these parameters on I/O performance. We run the preliminary evaluations in the NERSC Cori supercomputer. We provide detailed explanations for performance variations in our evaluations. In this abstract, we highlight three optimization strategies: **(i)** Accessing metadata blocks is a significant overhead for HDF5, which leads to low performance when the application reads one column. **(ii)** Reading and writing one file per chunk are the bottleneck for Zarr, especially when the chunk size is small. **(iii)** Memory copy slows HDF5 and Zarr when the array serialization in disk, decided by the chunk shape, does not match with the serialization in memory.

2 BENCHMARKS

We developed a benchmark for comparing array management libraries, based on a neuroscience dataset that is generated by 64 instruments observing the brain activity in 35,660,170 timesteps. The instrument flushes its data to the array per 69,649 timesteps. Hence, the array has 35,660,170 rows and 64 columns.

Applications typically access the entire array, and all observations in a set of timesteps or the observations of an instrument. Hence, the benchmark has three access patterns, which are reading or writing **(i)** an entire array, **(ii)** a set of rows, and **(iii)** a column.

3 EXPERIMENTAL RESULTS

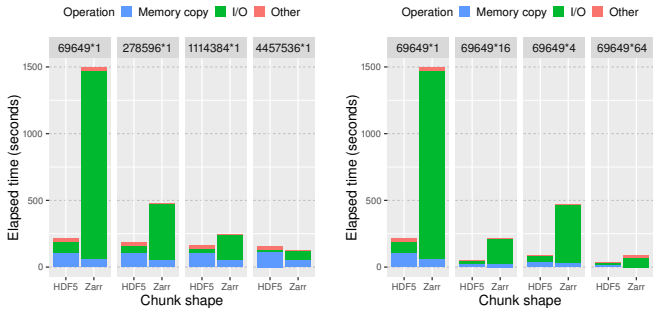
We systematically compare HDF5 and Zarr in different access patterns. Each experiment is repeated by 10 times and we report the median values.

3.1 Row-major read

In Figure 1, we show the time breakdown for reading an array with HDF5 and Zarr for various chunk dimensions. In this experiment, we increase the chunk size in both of the two dimensions. HDF5 and Zarr read the chunks in row-major order. The I/O time decreases as we increase the chunk size. HDF5 I/O time decreases from 78 seconds to 19 seconds and Zarr I/O time decreases from 1408 seconds to 68 seconds. Zarr has higher I/O cost because Zarr reads chunks from separate files while HDF5 reads from a single file. Opening files in Lustre has to get the file metadata from MDS, which is the bottleneck in the Zarr I/O process. The chunk size itself in this experiment decides the I/O time. Chunk shapes, having the same chunk size, result in the same I/O performance.

SC'19, November 17–22, 2019, Denver, Colorado, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn



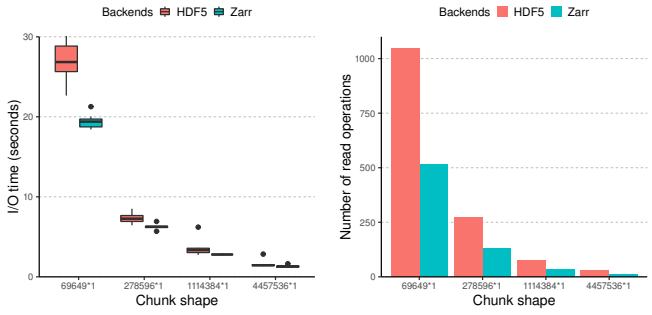
(a) Increase chunk length in the first dimension

(b) Increase chunk length in the second dimension

Figure 1: Time breakdown of reading an array

The cost for copying data from non-contiguous locations to a contiguous buffer is labeled as "Memory copy" in the figure.Memory copy time is constant as we increase the chunk length in the first dimension, while decreases as we increase the chunk length in the second dimension. Both HDF5 and Zarr read a chunk from Lustre to a temporary buffer, and then copy elements from the buffer to the output buffer where the array is serialized in row-major order. When we increase the chunk length in the first dimension, the number of copy operations does not change where an operation copies an element. However, as the chunk length in the second dimension increases, the number of elements copied in an operation is also increased, such that the number of copy operations decreases.

3.2 Column-major read



(a) I/O time

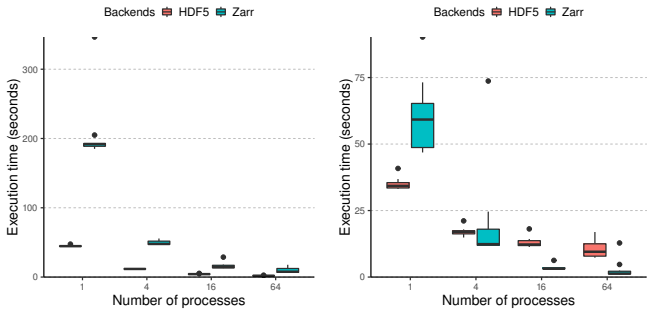
(b) Number of I/O operations

Figure 2: Read a column by HDF5 and Zarr.

In Figure 2, we show the I/O time and number of I/O operations when HDF5 and Zarr read a column. Both HDF5 and Zarr I/O time decreaes as we increase the chunk length in the first dimension. Reading a column with larger chunks in this experiment needs less I/O requests, where each I/O transfers more data.

Figure 2b shows that HDF5 has twice I/O operations than Zarr, such that HDF5 is slower shown in Figure 2a. Zarr only reads chunks, but HDF5 also reads metadata blocks. A metadata block describes the layout of fixed number of chunks which are contiguous in row-major serialization. Reading a single column requires a large amount of metadata blocks. Due to the extra metadata read operations, HDF5 has higher I/O time.

3.3 Parallel I/O



(a) Read arrays

(b) Write arrays

Figure 3: Execution time of parallel I/O

In this experiment, we use multiple processes concurrently and independently read and write arrays. The original array in the benchmark is partitioned to smaller ones, where each process accesses a small array. Figure 3 shows that the execution time of HDF5 and Zarr decreases as we increase the number of processes from 1 to 64. HDF5 is faster than Zarr in the read experiment, because Zarr has to open and read from multiple files. Lustre prefetches data for HDF5 processes. HDF5 is slower than Zarr in the write experiment, except for using 1 process to write the array. One possible reason is that parallel HDF5 writes metadata blocks. These blocks are scattered in the file, which leads to lots of random I/Os.

4 CONCLUSION

Array management libraries, together with MPI-IO and Lustre file system, are the common storage software stack used in HPC systems. This work derives a neuroscience benchmark to evaluate the performance of two array management libraries, HDF5 and Zarr. The benchmark contains a real scientific array and a set of access patterns. We study the impact of the chunk shape and the number of processes. In the preliminary results, increasing the chunk size and the number of processes improves the performance. Memory copy cost is high when the array serialization in disk and input or output buffer does not match. Accessing the metadata blocks is a bottleneck for HDF5 for parallel I/O. Zarr is slower than HDF5 when we read arrays in serial or parallel, because it opens and reads one file per chunk. The results and analysis provide the heuristics to select and tune array management libraries for optimal performance.

ACKNOWLEDGMENTS

This research was supported in part by NSF grants CCF-1816577.

REFERENCES

[1] Babak Behzad, Huong Vu Thanh Luu, Joseph Huchette, Surendra Byna, Ruth Aydt, Quincey Koziol, Marc Snir, et al. 2013. Taming parallel I/O complexity with auto-tuning. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 68.

[2] Oliver Ruebel, Andrew Tritt, Benjamin Dichter, Thomas Braun, Nicholas Cain, Nathan Clack, Thomas J Davidson, Max Dougherty, Jean-Christophe Fillion-Robin, Nile Graddis, et al. 2019. NWB: N 2.0: An Accessible Data Standard for Neurophysiology. (2019).

[3] Weikuan Yu, Jeffrey S Vetter, and H Sarp Oral. 2008. Performance characterization and optimization of parallel I/O on the Cray XT. In *2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 1–11.