Accelerating Parallel Write via Deeply Integrating Predictive Lossy Compression with HDF5

Sian Jin*, Dingwen Tao*, Houjun Tang†, Sheng Di‡, Suren Byna†, Zarija Lukic†, Franck Cappello‡
* Indiana University, Bloomington, IN, USA; Email: {ditao, sianjin}@iu.edu
† Lawrence Berkeley National Lab, Berkeley, CA, USA; Email: {htang4, sbyna, zarija}@lbl.gov
‡ Argonne National Laboratory, Lemont, IL, USA; Email: {sdi1, cappello}@anl.gov

***Abstract*—Lossy compression is one of the most efficient solutions to reduce storage overhead and improve I/O performance for HPC applications. However, existing parallel I/O libraries cannot fully utilize lossy compression to accelerate parallel write due to the lack of deep understanding on compression-write performance. To this end, we propose to deeply integrate predictive lossy compression with HDF5 to significantly improve the parallel-write performance. Specifically, we propose analytical models to predict the time of compression and parallel write before the actual compression to enable compression-write overlapping. We also introduce an extra space in the process to handle possible data overflows resulting from prediction uncertainty in compression ratios. Moreover, we propose an optimization to reorder the compression tasks to increase the overlapping efficiency. Experiments with up to 4,096 cores from Summit show that our solution improves the write performance by up to 4.5× and 2.9× over the non-compression and lossy compression solutions, respectively, with only 1.5% storage overhead (compared to original data) on two real-world HPC applications.**

***Index Terms*—parallel I/O, lossy compresion, HDF5**

I. Introduction

Large-scale scientific simulations on HPC systems play an important role in today's science and engineering domains. Such simulations can generate extremely large amounts of data that are highly compute and storage intensive. For example, one Nyx [1] cosmological simulation with a resolution of $4096 \times 4096 \times 4096$ cells can generate up to 2.8 TB of data for a single snapshot; a total of 2.8 PB of disk storage is needed, assuming the simulation runs 5 times with 200 snapshots dumped per simulation. Nowadays, the ever-increasing computation power can be utilized to run the simulations. However, managing such large amounts of data remains a major challenge. It is impractical to save all the generated raw data to disk due to: (1) the limited storage capacity even for supercomputers, and (2) the I/O bandwidth required to save these data can create bottlenecks in the transmission [2]–[5].

Lossy compression has been identified as one of the major data reduction techniques to address this issue. Specifically, a new generation of error-bounded lossy compression techniques, such as SZ [6]–[8], ZFP [9], MGARD [10] and their GPU versions [11]–[13], have been widely used in the scientific community [4], [6]–[9], [14]–[18]. Compared to lossless compression with up to 2× of compression ratio on scientific data [19], error-bounded lossy compressors provide much higher compression ratio with controllable loss of accuracy.

Dingwen Tao is the corresponding author.

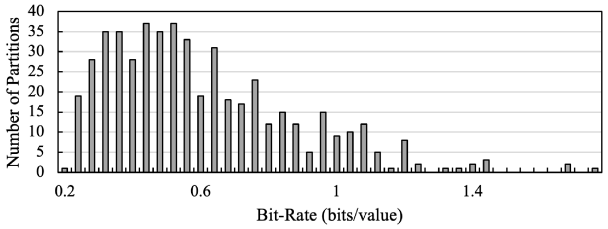


Fig. 1: Compression bit-rate distribution on a Nyx dataset with 512 partitions. Every partition uses the same compression configuration.

Scientific applications running on Supercomputers typically use parallel I/O libraries such as Hierarchical Data Format 5 (HDF5) [20] for managing their data. In specific, HDF5 is considered to provide high parallel I/O performance, portability of data, and rich APIs for managing data on these systems. It has been heavily used at supercomputing facilities for storing, reading, and querying scientific datasets [21], [22]. This is because HDF5 has specific designs and performance optimizations for popular parallel file systems such as Lustre [23], [24]. In addition, instead of using a general database in distributed storage, these datasets have their specific data management approach based on the parallel file system [23], [24]. Moreover, HDF5 also provides users dynamically loaded filters [25] such as lossless and lossy compression [26], which can automatically store and access data in compressed formats. Thus, it allows scientific applications to store and access the data in compressed formats. Parallel I/O in HDF5 with lossy compression filters can not only significantly reduce data size, but also improve the overall I/O performance.

However, the existing implementation of HDF5 with compression filters cannot fully utilize the benefits of reductions on data size and parallel-write time. This is because to write the data from different processes to a shared file, one must specify the offset of each data partition before writing the data. Thus, the parallel write cannot start until all compression tasks finish and the compressed sizes are communicated among all processes. Moreover, the compressed size of each data partition is distributed across a wide range of bit-rates (i.e., bits/value) and restricts any simple pre-allocation strategy. Figure 1 shows the distribution of compressed bit-rates on a Nyx dataset. When using lossless compression filters, the parallel-write performance could be even lower than the original non-

SC22, November 13-18, 2022, Dallas, Texas, USA
978-1-6654-5444-5/22/$31.00 ©2022 IEEE

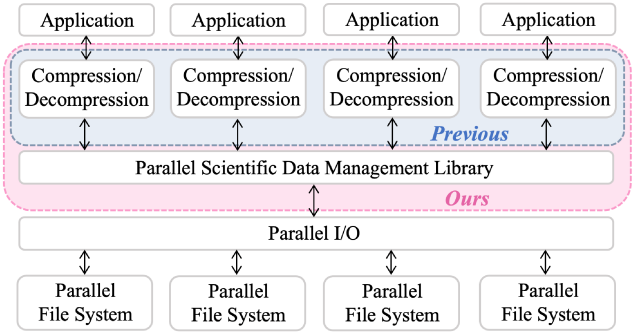


Fig. 2: Scientific data management with compression.

compression solution due to this high overhead [27].

To solve this issue, we propose a parallel write solution that integrates predictive lossy compression with the asynchronous I/O feature in HDF5, which overlaps I/O latency with compression to significantly improve the parallel write performance [23], [28]. Inspired by the previous research of Jin *et al.* [29] that estimates the compression ratio of prediction-based lossy compression with little overhead, we propose to predict the time of compression and parallel write before the actual compression, and leverage the asynchronous I/O feature to overlap compression with write. We also introduce an extra space to handle the uncertainty of the prediction. Moreover, we propose an optimization algorithm to reorder each process's compression tasks to increase the overlapping efficiency. The contributions of this paper are summarized as follows.

- We extend the prediction model for compression ratio to predict the throughputs of compression and parallel write for prediction-based lossy compression such as SZ.
- We propose a new compression-write scheme with HDF5 that can efficiently write the compressed data from different processes to a shared file by overlapping compression with write based on our prediction models.
- We optimize the execution order of compression tasks in each process to achieve higher parallel-write performance.
- We evaluate our proposed solution on two real-world HPC applications with up to 4,096 cores on Summit supercomputer and up to 512 cores on Bebop cluster. Experiments demonstrate that our solution improves the parallel-write performance by up to $4.5\times$ and $2.9\times$ compared to the HDF5 write without compression and with the SZ lossy compression filter (called "H5Z-SZ") [26], respectively, with only 1.5% storage overhead.

In Section II, we discuss the research background. In Section III, we present our design of parallel write with compression. In Section IV, we present our evaluation results. In Section V, we conclude our work and discuss future work.

II. Research Background and Challenges

A. Parallel I/O Libraries for HPC Applications

HPC applications generate and analyze massive amounts of data. A critical requirement of these applications is the capability to access and manage this data efficiently on HPC systems. Parallel I/O becomes the key technology to enable efficiently moving data between compute nodes and storage considering the complex storage hierarchy including node-local persistent memory, burst buffers, and disk-based storage. For example, HDF5 [20], netCDF [30], and Adaptable IO System (ADIOS) [31] are the most widely used high-performance I/O libraries for HPC applications. However, these I/O libraries still suffer from handling extremely large files (e.g. petabytes and beyond) due to inevitably limited I/O bandwidths. Therefore, compression techniques are often adopted by them [32]. For instance, H5Z-SZ [26] is a data filter for integrating SZ compression into HDF5.

Figure 2 shows the abstraction of different layers in the I/O systems, where compression is an individual layer in these systems. Specifically, compression is normally performed between generating and storing the data (e.g., H5Z-SZ). It is worth noting that the compression tasks of all processes and the parallel writing of compressed data to a shared file must be performed sequentially; in other words, there must be a synchronization between these two steps. This is because parallel writing of data from different processes to a shared file requires the size/offset of each data partition, but different processes may have vastly different data sizes after compression (even with the same size before compression). Thus, the current compression-write scheme cannot fully utilize the high compression ratio provided by lossy compression, especially for large-scale HPC applications with multiple data fields.

In this paper, considering HDF5 is well received by the HPC community as a system supporting parallel I/O, we mainly focus our performance evaluation on HDF5 without loss of generality. In addition, to improve the performance and productivity, a recent release of HDF5 [23] implements virtual object layer (VOL) which can redirect I/O operations into VOL connector and allow asynchronous I/O [28]. This feature enables an application to overlap I/O with other operations such as compression. Therefore, we can leverage this capability to deeply integrate and overlap predictive lossy compression with parallel write to improve the parallel write performance. Moreover, we focus on the parallel write to a large shared file due to three main factors: (1) it reduces scientists' workload to manage multiple files for storage, post-hoc analysis, and visualization; (2) it reduces the performance overhead of opening/closing multiple files and the storage overhead of metadata for many small files; and (3) partial processes (e.g., up to 4096 processes in [33]) of a large-scale simulation with subfiling still writes to a shared file.

B. Error-Bounded Lossy Compression

Lossy compression can compress data with extremely high compression ratio by losing non-critical information in the reconstructed data. Two types of most important metrics to evaluate the performance of lossy compression are: (1) compression ratio, i.e., the ratio between original data size and compressed data size, or bit-rate, i.e., the number of bits on average for each data point on average (e.g., 32/64 for single/double-precision floating-point data before compression); and (2) data distortion metrics such as peak signal-to-noise ratio (PSNR) to measure the reconstructed data qual-

ity compared to the original data. In recent years, a new generation of high accuracy lossy compressors for scientific data have been proposed and developed for scientific floating-point data, such as SZ [6]–[8] and ZFP [9]. These lossy compressors provide parameters that allow users to control the loss of information due to lossy compression precisely. Unlike traditional lossy compressors such as JPEG [34] which are designed for images (in integers), SZ and ZFP are designed to compress floating-point data and can provide a strict error-controlling scheme based on user's requirements. Generally, lossy compressors provide multiple compression modes, such as error-bounding mode. Error-bounding mode requires users to set an error type, such as point-wise absolute error bound and point-wise relative error bound, and a bound value (i.e., 10^{-3}). The compressor ensures that differences between original and reconstructed data do not exceed the error bound.

SZ is a prediction-based error-bounded lossy compressor for scientific data. It has three main steps: (1) predict each data point's value based on its neighboring points by using an adaptive, best-fit prediction method; (2) quantize the difference between the real value and predicted value based on the user-set error bound; and (3) apply a customized Huffman coding and lossless compression to achieve a high ratio.

Prior works have studied the impact of lossy compression on reconstructed data quality and post-hoc analysis, providing guidelines on how to set the compression configurations for certain applications [8], [17], [35]–[38]. For example, a comprehensive framework was established to perform a systematic analysis on compression configurations with a given dataset and provides the best-fit solution that satisfies the post-hoc analysis [18]. Moreover, Jin *et al.* [29] proposed a theoretical ratio-quality model to efficiently maximize the compression ratio given the quality constraints of post-hoc analysis. Note that similar to the previous work [39] on improving communication efficiency via lossy compression, this work assumes that the compression-configuration is set up by users based on their requirements of data quality, thus, the above compression-configuration methods are orthogonal to our solution.

C. Target I/O-Intensive HPC Applications

In this paper, we focus mainly on two I/O-intensive HPC applications—Nyx [22] and VPIC [40], which have been used for many previous I/O studies [41]–[46].

Nyx is an adaptive mesh, hydrodynamics code designed to model astrophysical reacting flows on HPC systems [1], [22]. This code models dark matter as discrete particles moving under the influence of gravity. The fluid in gas-dynamics is modeled using a finite-volume methodology on an adaptive set of 3-D Eulerian grids/meshes. The mesh structure is used to evolve both the fluid quantities and the particles via a particle-mesh method. For parallelization, Nyx uses MPI for the long-range force calculation and architecture-specific programming language for the short-range force algorithms, such as OpenMP and CUDA. Nyx data uses multiple 3-D arrays to represent field information in grid structure. According to prior studies [22], [47], it can run up to millions of cores on leadership supercomputers such as Summit [48].

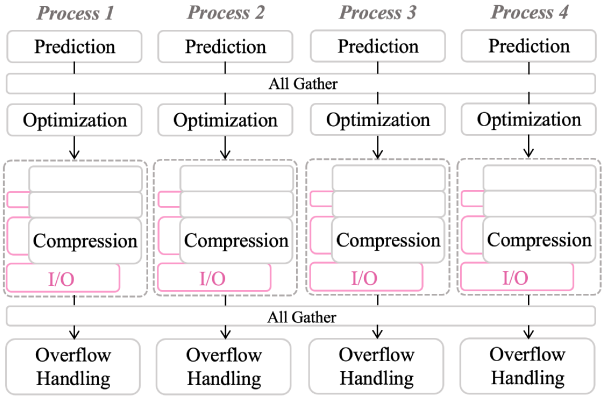


Fig. 3: Overview of our proposed solution.

VPIC (vector particle-in-cell) is a large-scale plasma physics simulation that can produce an unprecedented amount of data [40]. Collisionless magnetic reconnection is an important mechanism that releases explosive amounts of energy as field lines break and reconnect in plasmas. This reconnection also plays an important role in a variety of astrophysical applications that involve both hydrogen and electron-positron plasmas, including when the Earth's magnetosphere reacts to solar eruptions which can interfere satellite communication. Simulation of magnetic reconnection with VPIC is inherently a multi-scale problem that initiated in the small scale around individual electrons but eventually leads to a large-scale reconfiguration of the magnetic field. Recent simulations have revealed that electron kinetic physics is important not only in triggering reconnection, but also in its subsequent evolution [49]. This means plasma physics scientists find that they need to model the detailed electron motion, and that modeling poses severe computational challenges for 3D simulations of reconnection. A full-resolution magnetosphere simulation is an exascale-class computing problem.

III. Proposed Design Methodology

In this section, we present our proposed design that deeply integrates the prediction-based lossy compression into HDF5 to significantly improve the parallel-write performance.

A. Design Overview

Figure 3 shows an overview of our proposed framework. Specifically, we first conduct the ratio and throughput prediction phase for all data partitions on each process, which includes estimations of compression ratio, compression throughput, and I/O throughput. Then, an all-gather communication is performed to distribute the estimated compression ratio of each partition to all processes. Note that the estimations of compression throughput and I/O throughput are not distributed in this phase, since each process only requires these estimations of its own data partition in order to perform the following optimization. Next, each process computes the offset of its own data partition of each field for parallel write based on the estimated compressed data size with an extra

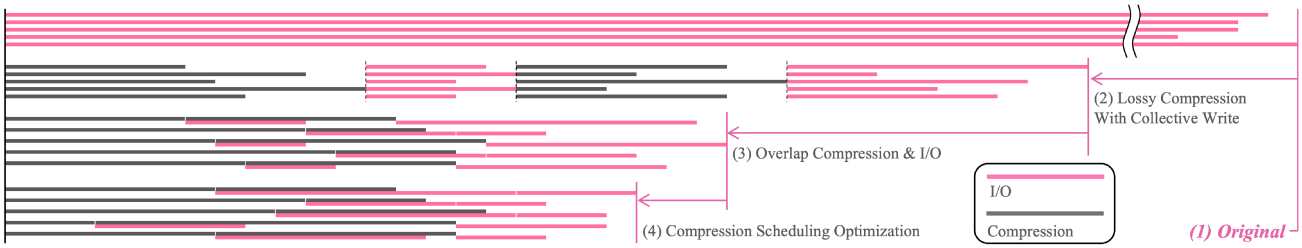


Fig. 4: Timeline of data aggregation with 5 processes and 2 data fields.

space (a mechanism to handle overflow data under unexpected prediction failure). Note that since each process gathers the same estimation results from others, the computation of offsets is consistent across all processes. After that, we perform an optimization on the order of compressing different data fields in each process. Lastly, we overlap compressions and writes based on the predicted write offsets and the optimized compression order (the order of compressing different fields).

Compared to the non-compression solution that writes the entire data to a shared file, our solution can significantly improve the overall write performance due to the high compression ratio of lossy compression that reduces I/O traffic. Compared to the previous lossy compression solution using HDF5 filter, our solution can overlap compression and write that allows independent and asynchronous write across processes. Figure 4 shows the simplified timeline of parallel write from five processes to a shared file on two data fields with four methods: (1) collective write without compression; (2) collective write with compression via filter; (3) independent writes overlapped with compressions; and (4) independent writes overlapped with reordered compressions.

Specifically, the improvement from method (2) to method (3) is due to the overlapping. From method (3) to method (4), the scheduling optimizer reordered some compression tasks in each process to further improve the overlapping efficiency. For example, the optimizer switches the compression order of the two data fields in the last process, i.e., the data with smaller compressed size are compressed later. The improvement of this optimization is more significant when more data fields are compressed in parallel write as it has higher chance to improve the overlapping efficiency. We will discuss more observations and insights in detail in Section IV-D.

In this paper, we focus on accelerating parallel write due to two main reasons: (1) HPC simulations are mostly write-oriented [50], and (2) parallel I/O libraries usually have lower performance in write than in read [23], [30], [51]–[53].

In the following sections, we first introduce the prediction models for prediction-based lossy compressors such as SZ. Next, we present the estimation of I/O throughput based on our models and empirical studies. After that, we propose our compression-write overlapping design with HDF5. Finally, we propose our algorithm for compression scheduling.

B. Compressor Throughput Estimation

The estimation on compression ratio in this work is based on recent research [29], which proposed a ratio-quality model

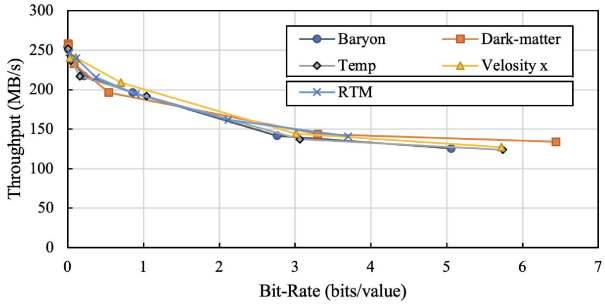


Fig. 5: Single-core compression throughput with different bit-rates on a Nyx and a RTM datasets. Evaluated on Bebop.

(i.e., compression ratio and reconstructed data quality model) for prediction-based lossy compressor. It developed algorithms to predict the compression ratio based on a newly designed sampling strategy without performing compression, significantly reducing the time overhead of the compression ratio prediction to less than 10% (in relative to the compression time). We leverage this idea to enable the overlapping design with small prediction overhead in our design.

On the other hand, estimating the compression throughput is also essential for compression order optimization. Generally, higher compression ratio results in higher compression throughput [17], due to the shorter time for building Huffman tree and encoding with smaller tree. However, there is no prior work for an accurate compression-throughput estimation.

In this work, we propose to estimate compression throughput based on the predicted compression ratio for each data partition. Figure 5 shows our empirical evaluation of compression throughput and compression ratio with multiple data fields and data types across different error bounds. Note that we use bit-rate (the average bits used to represent each value) on the x-axis to better illustrate the size of compressed data. For example, a bit-rate of 4 is equivalent to a compression ratio of 8× when compressing single-precision floating-point data (32 bits/value). The evaluation is performed on a single CPU core with SZ lossy compressor, since our target method (4) is to compress the data in each process/core independently before writing the compressed data to the shared file. The figure illustrates that (1) the maximum and minimum compression throughput are similarly bounded across different data samples (i.e., about 120∼250 MB/s); and (2) the bitrate-throughput curve for each data sample is highly consistent.

The reasons behind the first observation are: (1) For extremely high error bounds, despite the small size of the Huffman tree (negligible tree building time), the prediction

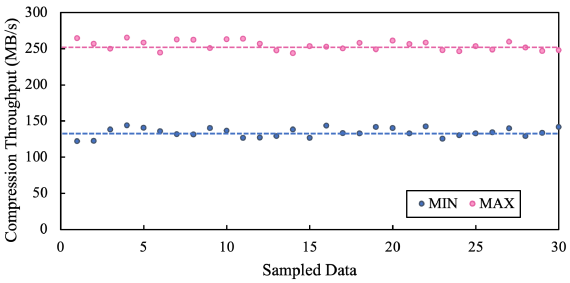


Fig. 6: Minimum and maximum compression throughput of a given data partition based on 30 samples from Baryon density, dark matter density, temperature and velocity x data fields in a Nyx dataset.

and encoding process still passes each point, which provides an upper bound on the throughput; (2) For extremely low error bounds, since SZ limits the maximum size of the Huffman tree (the maximum number of quantization codes), if one point is unpredictable with the maximum number of quantization codes, its original value will be saved directly, which provides a lower bound on the throughput. Figure 6 shows the maximum and minimum compression throughput evaluated on multiple data samples size of 67.1 MB from different data fields and types on the same experiment platform.

The reason behind the second observation is that for different data samples, the quantization codes after prediction are all centrally distributed, and the overall bit-rates are similar under the same Huffman encoding efficiency, where the sizes of the Huffman trees are also similar, resulting in stable tree building time and encoding time.

Based on Figure 5, we use a power function to predict compression throughput

$$T_{comp} = D/S \tag{1}$$
$$= (B_{ori} \times n)/(((C_{max} - C_{min}) \times 3^{-a})B^a + C_{min}),$$

where T_{comp} is the estimated compression time; D is the size of the original data; S is the compression throughput; B is the compressed bit-rate (i.e., the average bits used to save each value, e.g., a compression ratio of 16 from 32 bits floating point values results in $B = 2$); B_{ori} is the original bit-rate (i.e., 32 for single-precision floating point values); n is the number of points in the current data partition; C_{min} and C_{max} are the minimum and maximum compression throughputs, respectively; a is a hyper-parameter to describe the shape of the power function (i.e., a value smaller than 0; the lower the value, the more curved the function). The compression-throughput estimation is a power function that takes min-max into account. Note that the number 3 in the equation is based on our experiment that yields the best result.

This prediction model can be adapted for different machines, where we perform lossy compression on a sample of dataset offline to evaluate the minimum and maximum compression throughput and map it based on Equation (1). We show the accuracy of our proposed model in Section IV-B.

C. Write Time Estimation

The write time estimation is needed when optimizing the compression order, since the compressed data with larger estimated write time tends to be processed earlier to maximize the

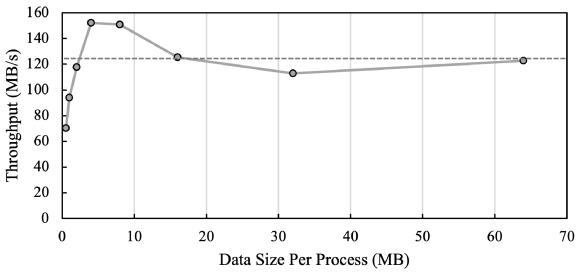


Fig. 7: Independent write I/O throughput per process with different data sizes per process. Evaluated on 128 processes.

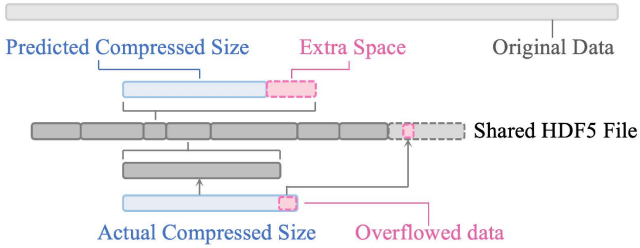


Fig. 8: Overflow data handling with preserved extra space.

I/O traffic occupancy. However, compared with the estimation on compression ratio and throughput, the accuracy required for write-time estimation is considerably lower. This is because one inaccurate write-time prediction makes a linear shift to all other write-time predictions within this process, so the relative time spent for each write remains unchanged compared to the entire time, and the optimization would not be affected (but only increase or decrease the actual write time). As a result, our goal is not to provide a highly accurate write-time estimation for each data partition, but to provide a capability to estimate the relative write time across different data sizes.

Figure 7 shows the parallel-write throughput per process with different data partition sizes. We can observe that the average throughput first increases as the data size increases and stabilizes after the data size reaches a certain point. Similar to the compression-throughput estimation, the write-time estimation is also based on the predicted bit-rate. Note that the compression throughput is based on the uncompressed data size. Considering that usually each process handles a similar amount of data during the simulation, the variance of the compression-time difference is limited to T_{max}/T_{min}. While the write throughput is based on the compressed data size, which can easily vary over $10\times$ across data partitions according to our evaluation. This means the write time is mainly dependent on the compression ratio assuming the write throughput is relatively stable across different processes without I/O congestion. Thus, we estimate the write time as

$$T_{write} = (B \times n)/C_{thr}, \tag{2}$$

where T_{write} is the write time, B is the compressed bit-rate; n is the number of points in the data partition; and C_{thr} is the stable write throughput based on an empirical evaluation.

D. Overlapping Compression and Write

When writing data from different processes to a shared file, the well-known I/O libraries such as HDF5, MPI-IO, and ADIOS require users to provide an offset for each data

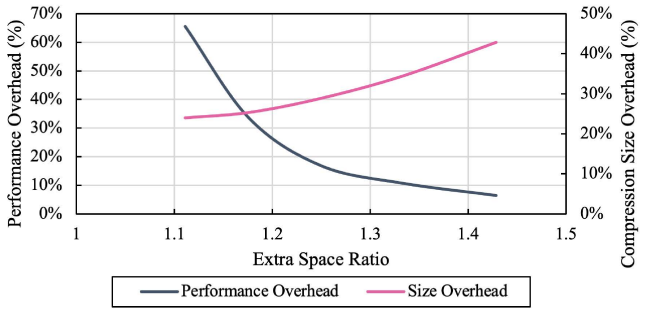


Fig. 9: Trade-off between performance overhead and compression size overhead. Shown the empirical average result based on Nyx and VPIC datasets on 512 processes.

partition. For the original non-compression parallel write, the offset of each partition can easily be computed based on the data size. However, with compression, the size of each compressed partition can vary drastically due to diverse data compressibility and different compression configurations. Prior to our work, it was impossible to pre-compute the offset before the compression, thus, it must sequentially process the collective write after compression. Thanks to the compression-ratio estimation for each partition [29], we can estimate the offset even before actual compression and allow overlapping between compression and write via HDF5's asynchronous I/O.

We note that although the compression-ratio model can provide high accuracy for prediction-based lossy compression, the prediction does not have a guarantee bound. This means that we must handle the situation when the actual compressed data size is larger than the predicted size, since it is impractical to simply recompute and update the offsets of all following partitions every time when the prediction failure happens as it would introduce a large amount of communication overhead. To this end, we propose to reserve an particular extra space for each prediction to handle the possibility of compressed data overflow. Moreover, we also propose an overflow data handling scheme to redirect and store the exceeded portion beyond the extra space. Figure 8 shows our proposed overflow handling strategy with an extra data space.

As the compression-ratio estimation error is not bounded, we cannot simply set a maximum extra space to guarantee no data overflow in practice. Thus, we take the extra space ratio as one of the tunable parameters in our framework. For example, setting this ratio R_{space} to 1.5 means that we over preserve 50% storage space (in relative to the predicted compressed size) when computing the offset of each partition.

In our evaluation, we find that the compression-ratio model [54], [55] performs poorly under extremely high compression ratio because of three factors: (1) when the compression ratio is high, the Huffman encoder can only provide a maximum compression ratio of $32\times$ (when the original data is single-precision) and relies on the following lossless compression to further reduce the encoded data; (2) the compression-ratio model is based on run-length encoding to analyze the lossless encoding efficiency, which naturally features lower estimation accuracy compared to the efficiency estimation of Huffman encoding; and (3) when the compression ratio is low, the encoded data stream from the Huffman encoder is highly random and is hardly further reduced by lossless compressors. Thus, when handling the data partition with compression ratio higher than 32 (bit-rate lower than 1), we set r_{space} to

$$r_{space} = \min(2, 1 + (R_{space} - 1) \times 4), \quad where \quad r_{comp} > 32. \tag{3}$$

where r_{comp} is the compression ratio. On one hand, the higher the extra space ratio is, the larger the storage size of the compressed shared file will be, due to the larger extra space that is likely to be wasted. On the other hand, increasing r_{space} may improve the overall performance due to less data overflow processing overhead, and vice versa. Overall, above a certain threshold (e.g., $r_{space} > 1.1\times$ in our evaluation), increasing r_{space} becomes a trade-off between the write performance and the overall compression efficiency.

More specifically, below this threshold, decreasing r_{space} results in a significant write performance drop due to a large number of compressed data overflows. For example, when $r_{space} = 1.1\times$, 32.4% data partitions suffer from data overflow, causing an extra 65.6% time overhead. Based on our observations from Figure 14, the trade-off between write performance and compression efficiency is relatively similar across different fields and datasets. We will demonstrate this with more details in Section IV-C.

As a result, we propose to weight the write performance and compression efficiency overhead and provide a tunable ratio between them for users. For any given weights, we provide the extra space ratio based on our proposed mapping. Figure 9 shows our mapping solution for the extra space ratio (a more detailed evaluation illustrating the accuracy of this mapping will be shown in Section IV-C). Note that we only support r_{space} between $[1.1, 1.43]$ due to (1) an extremely high time overhead below 1.1, and (2) a low efficiency of trading storage for performance after 1.43. We set the default extra space ratio to 1.25 in this work.

After estimating the offset of each data partition, we must store this offset information as metadata for the decompression purpose. Compared to the compressed size of the entire dataset, this metadata size is totally negligible. For example, when writing a total size of 2.5 TB Nyx dataset from 4,096 processes with SZ, there is only 295 KB metadata to store the offsets for 9 fields, while the compressed data size is 210 GB.

For handling the compressed data overflow, our goal is to append the exceeded data to the end of the shared file. Specifically, we continue to optimize and write the maximum amount of data into the preserved area of the shared file. Then, after all processes finish writing, we initiate an all-gather operation across all processes to distribute the size information of overflow data. After that, some processes calculate the offset of their own excess and write it to the end of the shared file independently. Note that since only a small fraction of processes may need to process/save a small amount of overflow data, the overall time overhead for processing overflow data is small. A more detailed evaluation will be presented in Section IV-D.

Algorithm 1 Compression Order Optimization

Notation: data fields in current process: ℓ; compression queue: Q; compression queue after insert and additional data: Q°; possible insert locations in a queue: β; time to compress: t_c; time to write:t_w; predicted compression time: $P_c(\ell)$; predicted write time: $P_w(\ell)$
Global: $P_c(\ell), P_w(\ell)$

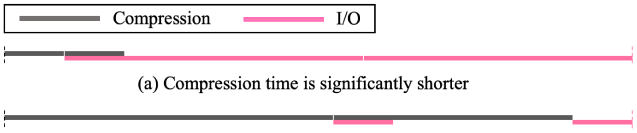
```
1  procedure TIME(q)
2      t_c, t_w ← 0
3      for ℓ ← data fields in q do
4          t_c ← t_c + P_c(ℓ)
5          t_w ← P_w(ℓ) + max(t_c, t_w)
6      end for
7      return t_w
8  end procedure
9
10 procedure SCHEDULINGOPTIMIZATOR
11     for ℓ ← data fields in current process do
12         for β ← all possible insert location do
13             Q° ← insert ℓ to β
14             if TIME(Q°) < TIME(Q) or first β then
15                 Q ← Q°
16             end if
17         end for
18     end for
19     return Q
20 end procedure
```

E. Compression Order Optimization

As shown in Figure 4, when writing data to the shared file, overlapping the compression and I/O can provide a significant performance benefit over the previous collective-write solution with lossy compression. However, with multiple data fields in each process, compressing them sequentially in the original order is not the optimal solution. In fact, since we use the modeling approach to predict the compressed size, the compression time, and the write time of each data partition, we can reorder the compression tasks for different fields to maximize the overlapping without any penalty.

Algorithm 1 shows the pseudocode for optimizing the compression order in each process. The proposed method is based on the observation that the total compression time is theoretically fixed regardless of the compression order. Our optimization focuses on the dependencies and timing of launching write operations for each compressed data to minimize timeouts compared to compression. The time complexity of the proposed algorithm is $O(n^2)$, whereas the time complexity of our compression is $O(N)$. Considering that N (i.e., the number of values) in one data partition is significantly larger than n (i.e., the number of data fields), the optimization overhead is almost negligible compared to the actual compression and write time. For example, this optimization overhead is only 0.17% of the compression time under an extreme condition where N is small at 32768 ($32 \times 32 \times 32$) but n is very large at 100.

Based on our algorithm design, we can expect the optimization to bring benefit when the balance between compression time and I/O time is relatively stable. However, we notice that in the extreme scenarios, the compression time and the write time could be very unbalanced, which can diminish the benefit of our reordering optimization, as shown in Figure 10. Specifically, there are two cases: (a) when the write time is significantly longer than the compression time, or (b) when the compression time is significantly longer than the write time. In both cases, there is no significant room to improve performance because of the limited overlap between compression and I/O (more details will be discussed in Section IV-D).



(a) Compression time is significantly shorter

(b) Compression time is significantly longer

Fig. 10: An example of extremely unbalanced compression time and write time, limiting the benefit from our reordering.

TABLE I: Details of Tested Datasets

Name	Description	Scale	Size
nyx [22]	Cosmology simulation	4096×4096×4096	2.47 TB
		2048×2048×2048	206.15 GB
		1024×1024×1024	25.76 GB
		512×512×512	3.22 GB
VPIC [56]	Particle simulation	161,297,451,573	4.62 TB

In addition, we note that when the number of data fields is relatively large, our optimization can bring greater benefit. This is because the overall performance is dependent on the worst process with the longest time among all the processes due to independent asynchronous writes.

IV. Experimental Evaluation

In this section, we present the evaluation results of our proposed framework for accelerating parallel write. We first evaluate the accuracy of our prediction for compression time and write time. Next, we evaluate the extra space setup under different compression ratios and scales. Finally, we perform performance evaluation and scaling study of our approach and compare it with the original solution without compression and the solution using using the H5Z-SZ filter [26].

A. Evaluation Setup

We rigorously implement our approach with HDF5 [23] and SZ3 [36] (a modularized prediction-based lossy compressor). We conduct our experiments on two HPC systems: (1) the Summit supercomputer [48] at Oak Ridge National Laboratory, each node of which is equipped with two IBM POWER9 processors with 42 physical cores and 512 GB DDR4 memory, and (2) the Bebop cluster [57] at Argonne National Laboratory, each node of which is equipped with two 18-core Intel Xeon E5-2695v4 CPUs and 128 GB DDR4 memory.

We use different scales of Nyx and VPIC datasets in our evaluation. Table I shows the details of our tested datasets. According to the previous work [17], [35], using the absolute error bounds of $(0.2, 0.4, 1e+3, 2e+5, 2e+5, 2e+5)$ for compressing the six Nyx data fields (i.e., baryon density, dark matter density, temperature, velocity x, velocity y, velocity z) can satisfy the post-hoc analysis quality with an average PSNR (peak signal-to-noise ratio) at 78.6 dB, resulting in a compression ratio of $\sim$16×. The $4096 \times 4096 \times 4096$ Nyx dataset has three additional fields: particle_vx, particle_vy and

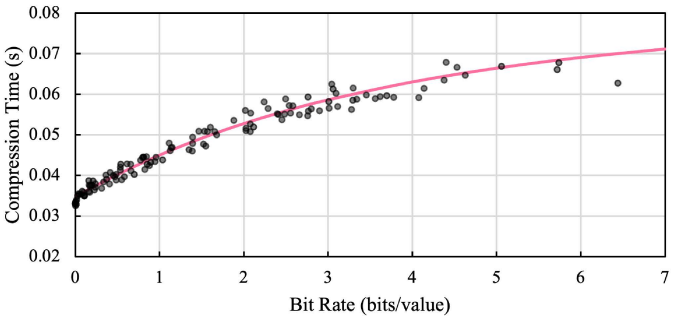


Fig. 11: Accuracy of our compression-time estimation on 512^3 Nyx data samples (red line is predicted time; black dots are actual time).

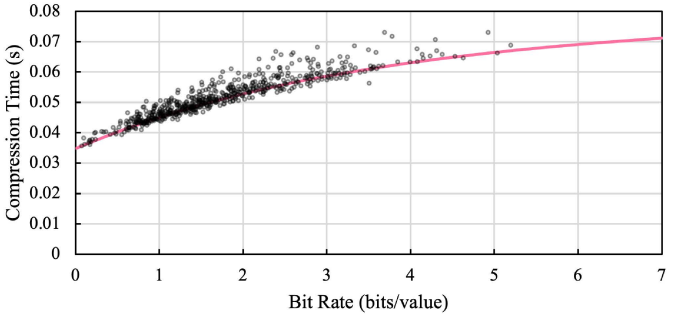


Fig. 12: Accuracy of our compression-time estimation on 1024^3 Nyx data samples. Red line is predicted time; black dots are actual time. Offline parameter is based on the baryon density of 512^3 Nyx dataset.

particle_vz. Similarly, we compress them with a compression ratio of $16\times$ to satisfy the post-hoc analysis quality. For the VPIC dataset, we also compress the 8 data fields with a compression ratio of $13.8\times$, which is suggested by the application developers according to their post-hoc analysis. Note that our solution can also be applied to other scientific datasets with similar performance improvement expected. This is because: (1) our solution provides the same reconstructed data quality compared to previous solutions, where many studies [8], [17], [18], [35]–[38] show that lossy compression has a high data reduction capability on a variety of applications; (2) a previous study shows that the accuracy of the compression-ratio estimation used in our solution is consistently above 90% across tens of benchmark datasets [29], implying that the proportion of data overflows due to inaccurately estimated compression ratios is relatively stable in most scientific datasets; (3) the accuracy of the compression-throughput estimation is consistently high across different datasets and fields, as shown in Figure 5, 11 and 12; and (4) the accuracy of I/O-throughput estimation only relies on the estimated compressed data size as discussed in (2).

B. Accuracy of Compression and I/O Throughput Estimation

First, we evaluate the accuracy of our proposed estimation on compression time. We perform the evaluation on the 512^3 Nyx dataset only using the baryon density field and conduct the offline compression with the relative error bound ranging between [1e-1, 1e-8]. Then we calculate C_{min}, C_{max} and a in Equation (1) based on the offline results, so that we can use Equation (1) to predict the compression time of any given

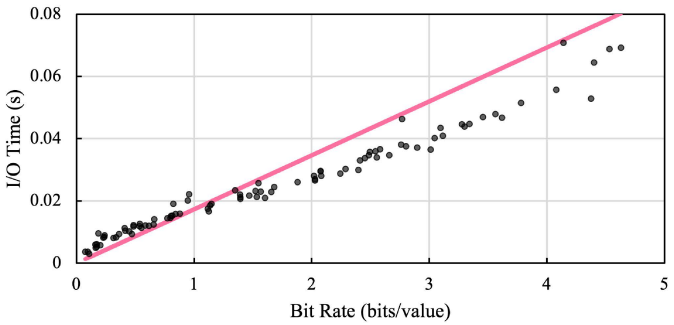


Fig. 13: Accuracy of our write time estimation on 1024^3 Nyx data samples. Red line is predicted time; black dots are actual time.

data (i.e., 101.7, 240.6, and -1.716, respectively, in this case). Next, we distribute the Nyx dataset to 64 processes (each process has a $128 \times 128 \times 128$ data partition) and perform the write operation. For each field of data partition in each process, we predict the compression time using Equation (1). Figure 11 shows the actual compression time versus the predicted compression time.

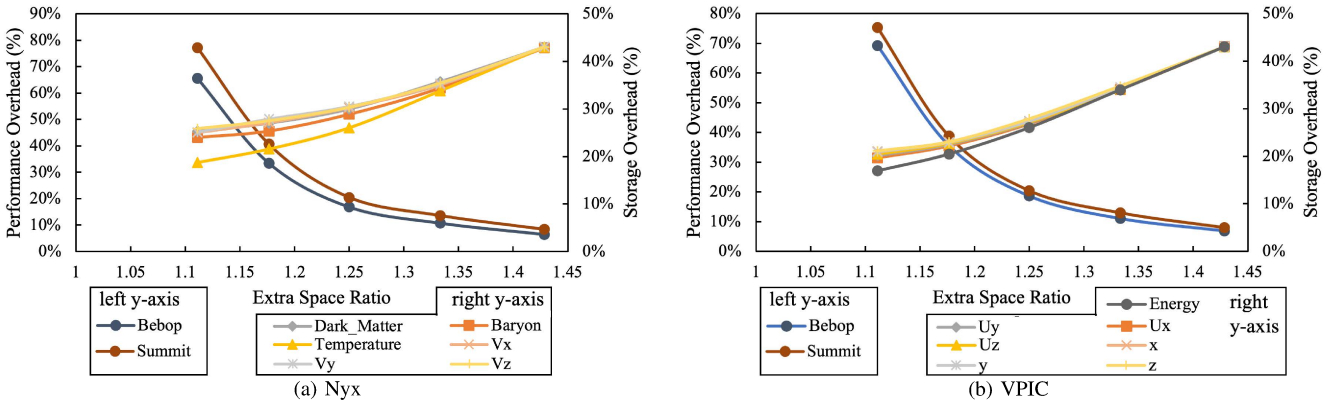
Furthermore, we extend our evaluation to the 1024^3 Nyx dataset and 512 processes using the same C_{min}, C_{max}, and a obtained from the 512^3 dataset. Figure 12 shows the actual compression time versus the predicted compression time. We can observe from both figures that our compression-time estimation has high accuracy even though the offline experiment only uses baryon density as input. This is consistent with the result shown in Figure 5, where different data fields and datasets have similar compression throughputs.

Similarly, we evaluate the accuracy of our proposed write-time estimation. We perform the offline evaluation by writing data from 128 processes into a shared file multiple times. The size of the data per process is set to 5 MBs, 10 MBs, 20 MBs, 50 MBs, or 100 MBs. We measure the average write throughput. Based on our observation, further increasing or decreasing the scale would not significantly affect the average I/O throughput per process; therefore, we only perform the offline evaluation on one scale with different data sizes to reduce the offline evaluation overhead. Next, we distribute the 1024^3 Nyx dataset to 64 processes, perform the write operation, and measure the time of independent write for each process along with the bit-rate after compression.

Figure 13 shows the comparison between the estimated write time and the actual write time. Note that the accuracy of low bit-rate is slightly lower than that of high bit-rate. This is because the compressed data size is very small (i.e., 0.94 MB in this case) under low bit-rate, which result in a significantly low write throughput. However, since we optimize the compression order based only on the absolute write time, such a small amount of write-time prediction error has hardly any effects on optimization decisions.

C. Evaluation on Extra Space Ratio

As mentioned in Section III-D, the extra space ratio is one of the most important parameters that balance the overall write performance and storage overhead in our framework. We then



(a) Nyx (b) VPIC

Fig. 14: Trade-off between performance overhead and storage overhead based on different extra space ratios on Nyx dataset (6 data fields) and VPIC dataset (7 data fields) on both Bebop and Summit with 512 processes. The target compressed bit-rate is 2 bits/value.

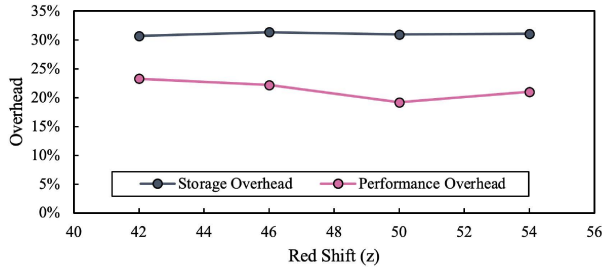


Fig. 15: Evaluation on the consistency of the storage and performance overheads using the same extra space ratio of 1.25 with 512 processes on Summit. Red shift stands for different time-steps (higher values means earlier time in the simulation).

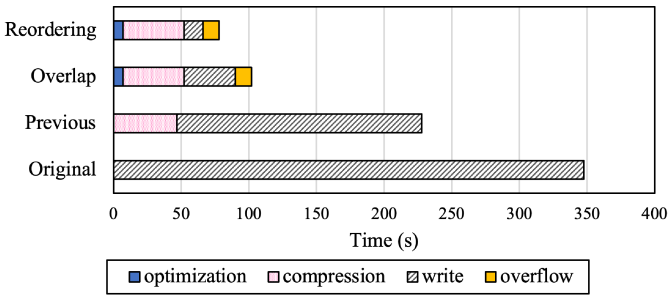


Fig. 16: Performance comparison among our solution (overlapping and reordering), original non-compression solution, and previous compression-write solution on 4096^3 Nyx dataset with 512 processes.

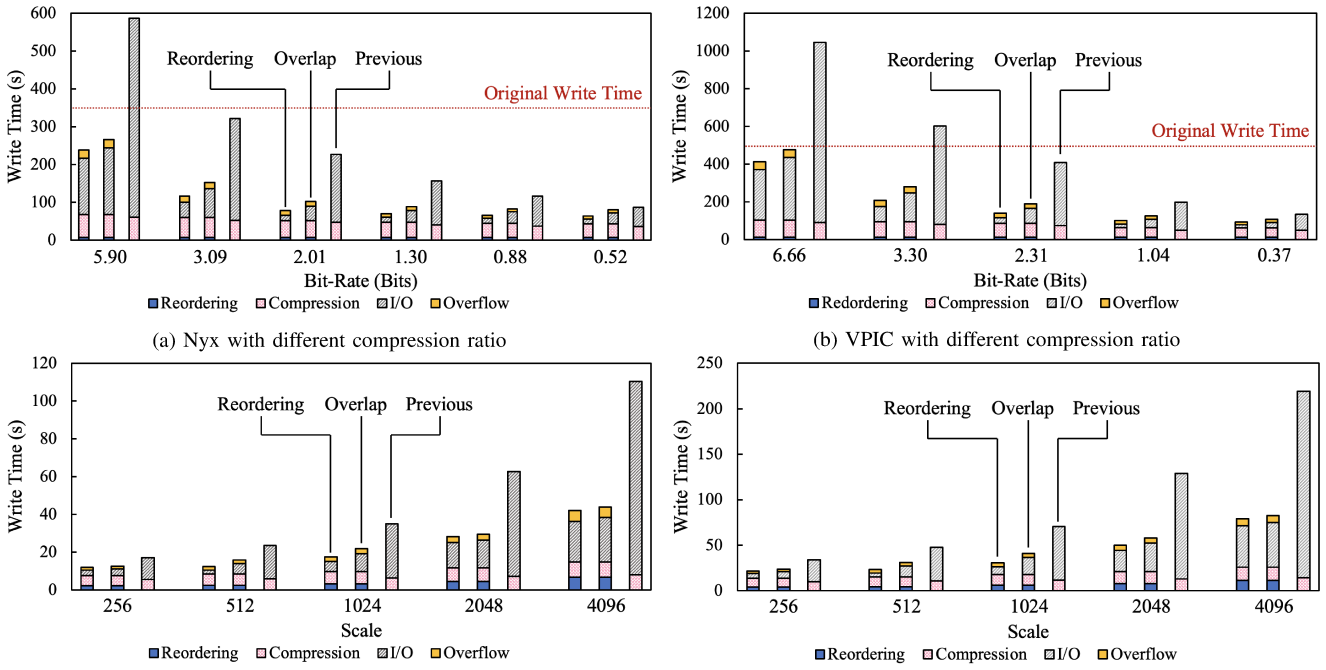
evaluate this performance-storage trade-off on both the Nyx and VPIC datasets. For the Nyx dataset, we use the 1024^3 dataset and distribute it to 512 processes for parallel write. For the VPIC dataset, we downscale the original dataset by sampling 10% data points and distribute it to 512 processes.

Figure 14 shows the correlation between the write-performance overhead and the storage overhead with the two datasets on both Bebop and Summit. Note that the write-performance overhead is compared with the write time without handling data overflow (excluding the compression time), and the storage overhead is compared with the ideal compressed data size (without the extra space). We can observe that for each dataset on a given system, the trade-off curve is highly similar across different data fields, since the accuracy of the compression-ratio model is relatively stable on the same dataset. We also note that even though the write-performance overhead and storage overhead with the same extra space ratio are different on the two datasets, the trade-off between the two overheads is very similar. This is because the lower upper bound of the compression-ratio estimation results in larger number of processes to hold overflow data and hence higher performance overhead, which also enlarges the total amount of overflow data and increases the overall storage overhead.

Regarding the results across the two systems, the difference of the performance overhead is mainly due to the higher I/O bandwidth of Summit over Bebop, which reduces the overall write time and enlarges the relative performance overhead. However, the performance-storage tread-off is still similar across different systems, which means that we can use our offline study to guide the online decision making for the trade-off between performance and storage. Figure 14 shows the averages of the performance and storage overhead in different extra space ratios across the four dataset-system setups (as mentioned in Section III-D). By default, we use the extra space ratio of 1.25 to minimize performance overhead while keeping low storage cost. In addition, we apply the extra space ratio of 1.25 with multiple time-steps in a series of Nyx datasets, as shown in Figure 15. It shows that our solution has consistent storage and performance overhead across time-steps. Another alternative approach is to sample the dataset from the first time-step and provide users a more accurate estimation on the trade-off between performance and storage. We will design a more user-friendly parameter-tuning mechanism in future.

D. Overall Performance Improvement

Next, we compare the performance of our proposed solution with the non-compression solution and the previous compression-filter solution. For the non-compression solution, the distributed data is written to the shared file without any compression. We implement it using HDF5 with independent write, which can significantly increase the performance over collective write [23]. For the previous compression-filter solution, the compression process is considered as a filter, and the parallel write starts only when every process finishes its compression on all data partitions and the compressed data sizes are known to all processes to compute the write offsets.



(a) Nyx with different compression ratio

(b) VPIC with different compression ratio

(c) Nyx with different scale

(d) VPIC with different scale

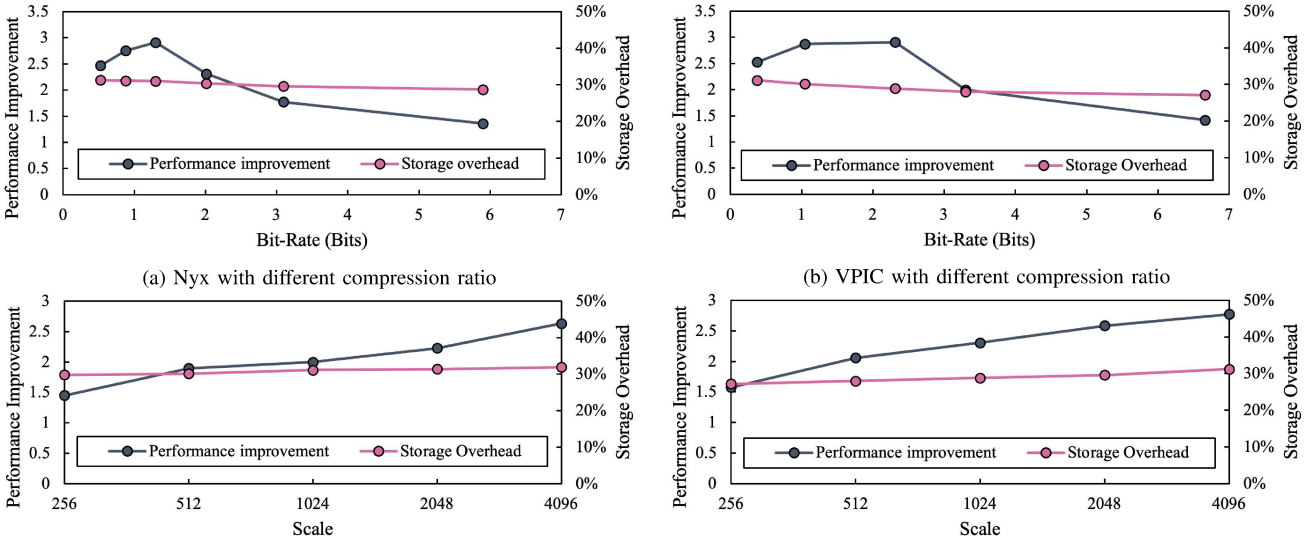
Fig. 17: Performance improvement of overall parallel-write with our proposed solution compared to the previous write solution with H5Z-SZ on both Nyx and VPIC datasets. Dashed red line is the baseline of HDF5 without compression. (a) and (b) are evaluated with 512 processes on Summit. (c) and (d) are evaluated with a target bit-rate of 2.

We implement it using HDF5 with collective write, since compression filters do not support independent write. Note that HDF5 only provides additional features such as asynchronous I/O and does not change the collective write.

Figure 16 shows the comparison of the performance breakdown between our proposed solution and the two existing solutions. We implement our solution both with and without the compression reordering, referred to as "overlapping" and "reordering" in the figure, respectively. Note that the write time bar shown in the overlapping solution is measured by the time between the end of the slowest compression and the end of the writing process rather than the entire write time (due to overlapping). The original/ideal compression ratio (without the extra space) in this experiment is $17.94\times$, while the actual compression ratio considering the extra space (i.e., the extra space ratio of 1.25) is $14.13\times$.

We can observe that the collective write solution with lossy compression still outperforms the independent write without compression by $1.87\times$ due to the high compression ratio and reduced data size. Compared to the previous compression-write solution, our overlapping solution further improves the overall write performance by $1.79\times$ due to the asynchronous independent write. Note that the compression times spent in the two solutions are similar, indicating that our framework improves the writing efficiency rather than the compression throughput. Furthermore, with compression order optimization, we further improve the performance by $1.30\times$ in addition to our overlapping optimization. The extra write time (gray bar) is significantly reduced due to the high overlapping efficiency. Overall, comparing our optimized solution to the non-compression write, we improve the write performance by $4.46\times$ with the compression ratio of $14.1\times$; compared to the previous write with H5Z-SZ with the compression ratio of $17.9\times$, our optimized solution improves the write performance by $2.91\times$ with a 26% storage overhead. Note that this overhead is compared with the compressed data size, if compared with the original data size, the storage overhead is equivalent to 1.5%.

Furthermore, we evaluate our framework with different overall compression ratios and different scales to illustrate the efficiency of compression reordering on both the Nyx and VPIC datasets. Figure 17a and Figure 17b show that the compression reordering optimization has a poor improvement over the overlapping-only solution under extremely high and low compression ratio, the corresponding performance improvement and storage overhead are shown in Figure 18a and Figure 18b. For extremely high compression ratio, the compressed data size in each process is tiny, so the write time is significantly smaller than the compression time, as shown in Figure 17a and Figure 17b (the short gray bar versus the orange bar). In this case, reordering the compression tasks provides little performance benefit, since the overall compression time cannot be further improved and the potential of reducing the extra write time is negligible. For extremely low compression ratio, the compressed data size in each process is large, so the overall compression time is significantly shorter than the write time, where the overlapping efficiency is likely to be sufficiently high even without optimization (low potential to optimize). Both datasets have a similar compression ratio ($14.1\times$ and $10.9\times$ for Nyx and VPIC, respectively) that



(a) Nyx with different compression ratio

(b) VPIC with different compression ratio

(c) Nyx with different scale

(d) VPIC with different scale

Fig. 18: Performance improvement (overall) and storage overhead of our solution compared to the previous solution on both Nyx and VPIC datasets. (a) and (b) are evaluated with 512 processes om Summit. (c) and (d) are evaluated with a target bit-rate of 2.

benefits the compression reordering optimization. Note that although the higher compression ratio almost always indicates the better write performance, it also means lower data quality with information loss that may potentially hurt post-hoc analysis. Under the extreme cases when the compression ratio is very small, the previous write solution with H5Z-SZ shows even worse performance than the non-compression write.

Finally, we perform the scaling study of our proposed framework. Specifically, we conduct a weak scaling study that keeps the same data partition size in each process, i.e., 256^3 and $39, 379, 260$ for the Nyx and the VPIC datasets, respectively. Figure 17c and Figure 17d illustrate that the compression order optimization over the overlapping solution is relatively stable on the evaluated scales at $(256, 512, 1024, 2048, 4096)$, the corresponding performance improvement and storage overhead are shown in Figure 18c and Figure 18d. This is because the overall compression time and write time in each process are mostly stable through different scales. However, a larger scale slightly benefits our solution because the independent, asynchronous write typically provides better scalability compared to the collective write used by the previous compression-write solution [23]. Note that the optimization time and the overflow time increase with the scale, because even though the prediction time is stable, larger scale introduces longer communication time for the all-gather operation.

In conclusion, under the circumstances of satisfying the user-set compression configuration, our parallel write with lossy compression is scalable and high-performance, especially when (1) the number of data fields is relatively large; (2) the overall compression ratio is preferably between $10\times$ and $20\times$ (with balanced compression time and write time); and (3) the data amount in each process is not too small (deserving compression). In addition, users can fine-tune the extra space ratio to make a good performance-storage trade-off.

V. Conclusion and Future Work

In this paper, we propose to integrate predictive lossy compression deeply with HDF5 to significantly improve the parallel-write performance for large-scale scientific simulations. We introduce a newly designed prediction method to estimate the compression and write time for each process and use this information to pre-compute the write offset before the actual compression. Furthermore, we introduce an extra space design to handle the uncertainty of the prediction and an optimization algorithm to reorder the compression tasks in each process. We evaluate our proposed solution on both Bebop cluster and Summit supercomputer with up to 4,096 cores. The evaluation shows that our solution can provide a $4.46\times$ performance improvement with a $14.1\times$ compression ratio compared to the original parallel write without compression, and provide a $2.91\times$ performance improvement compared to the previous parallel write with the SZ compression filter with only 20% compression ratio degradation.

In the future, we will extend our solution to other parallel I/O libraries such as ADIOS [31] and support more lossy compressors such as ZFP [9]. Moreover, we plan to evaluate our solution on more real-world HPC datasets.

Acknowledgments

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations—the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, to support the nation's exascale computing imperative. The material was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (ASCR), under contracts DE-AC02-06CH11357 and DE-AC02-05CH11231. This work was also supported by the National Science Foundation under Grants 2003709, 2042084, 2104023, 2104024, 2211538, and 2211539. We gratefully acknowledge the computing resources provided on Argonne's Bebop cluster and Oak Ridge's Summit supercomputer.

REFERENCES

[1] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Lukić, and E. Van Andel, "Nyx: A massively parallel amr code for computational cosmology," *The Astrophysical Journal*, vol. 765, no. 1, p. 39, 2013.

[2] L. Wan, M. Wolf, F. Wang, J. Y. Choi, G. Ostrouchov, and S. Klasky, "Comprehensive measurement and analysis of the user-perceived i/o performance in a production leadership-class storage system," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 1022–1031.

[3] ——, "Analysis and modeling of the end-to-end i/o performance on olcf's titan supercomputer," in *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2017, pp. 1–9.

[4] F. Cappello, S. Di, S. Li, X. Liang, A. M. Gok, D. Tao, C. H. Yoon, X.-C. Wu, Y. Alexeev, and F. T. Chong, "Use cases of lossy compression for floating-point data in scientific data sets," *The International Journal of High Performance Computing Applications*, 2019.

[5] C. Zhang, S. Jin, T. Geng, J. Tian, A. Li, and D. Tao, "Ceaz: accelerating parallel i/o via hardware-algorithm co-designed adaptive lossy compression," in *Proceedings of the 36th ACM International Conference on Supercomputing*, 2022, pp. 1–13.

[6] D. Tao, S. Di, Z. Chen, and F. Cappello, "Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization," in *2017 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2017, pp. 1129–1139.

[7] S. Di and F. Cappello, "Fast error-bounded lossy HPC data compression with SZ," in *2016 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2016, pp. 730–739.

[8] X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo, Z. Chen, and F. Cappello, "Error-controlled lossy compression optimized for high compression ratios of scientific datasets," in *2018 IEEE International Conference on Big Data*. IEEE, 2018, pp. 438–447.

[9] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.

[10] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky, "Multilevel techniques for compression and reduction of scientific data—the univariate case," *Computing and Visualization in Science*, vol. 19, no. 5–6, pp. 65–76, 2018.

[11] J. Tian, S. Di, K. Zhao, C. Rivera, M. H. Fulp, R. Underwood, S. Jin, X. Liang, J. Calhoun, D. Tao *et al.*, "cusz: An efficient gpu-based error-bounded lossy compression framework for scientific data," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 2020, pp. 3–15.

[12] J. Tian, S. Di, X. Yu, C. Rivera, K. Zhao, S. Jin, Y. Feng, X. Liang, D. Tao, and F. Cappello, "Optimizing error-bounded lossy compression for scientific data on gpus," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2021, pp. 283–293.

[13] cuzfp. [Online]. Available: https://github.com/LLNL/zfp/tree/develop/src/cuda_zfp

[14] T. Lu, Q. Liu, X. He, H. Luo, E. Suchyta, J. Choi, N. Podhorszki, S. Klasky, M. Wolf, T. Liu, and Z. Qiao, "Understanding and modeling lossy compression schemes on HPC scientific data," in *2018 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2018, pp. 348–357.

[15] H. Luo, D. Huang, Q. Liu, Z. Qiao, H. Jiang, J. Bi, H. Yuan, M. Zhou, J. Wang, and Z. Qin, "Identifying latent reduced models to precondition lossy compression," in *2019 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2019.

[16] D. Tao, S. Di, X. Liang, Z. Chen, and F. Cappello, "Optimizing lossy compression rate-distortion from automatic online selection between sz and zfp," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 8, pp. 1857–1871, 2019.

[17] S. Jin, P. Grosset, C. M. Biwer, J. Pulido, J. Tian, D. Tao, and J. Ahrens, "Understanding gpu-based lossy compression for extreme-scale cosmological simulations," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 105–115.

[18] P. Grosset, C. Biwer, J. Pulido, A. Mohan, A. Biswas, J. Patchett, T. Turton, D. Rogers, D. Livescu, and J. Ahrens, "Foresight: analysis that matters for data reduction," in *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 2020, pp. 1171–1185.

[19] S. W. Son, Z. Chen, W. Hendrix, A. Agrawal, W.-k. Liao, and A. Choudhary, "Data compression for the exascale computing era-survey," *Supercomputing Frontiers and Innovations*, vol. 1, no. 2, pp. 76–88, 2014.

[20] The HDF Group. Hierarchical data format version 5. [Online]. Available: http://www.hdfgroup.org/HDF5

[21] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, "An overview of the hdf5 technology suite and its applications," in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, 2011, pp. 36–47.

[22] Nyx. [Online]. Available: https://github.com/AMReX-Astro/Nyx

[23] S. Byna, M. S. Breitenfeld, B. Dong, Q. Koziol, E. Pourmal, D. Robinson, J. Soumagne, H. Tang, V. Vishwanath, and R. Warren, "Exahdf5: delivering efficient parallel i/o on exascale computing systems," *Journal of Computer Science and Technology*, vol. 35, no. 1, pp. 145–160, 2020.

[24] S. Pokhrel, M. Rodriguez, A. Samimi, G. Heber, and J. J. Simpson, "Parallel i/o for 3-d global fdtd earth–ionosphere waveguide models at resolutions on the order of˜ 1 km and higher using hdf5," *IEEE Transactions on Antennas and Propagation*, vol. 66, no. 7, pp. 3548–3555, 2018.

[25] HDF Group and others, "Hierarchical data format version 5, filter," 2000.

[26] H5Z-SZ. [Online]. Available: https://github.com/disheng222/H5Z-SZ

[27] The HDF5 team. Parallel compression improvements in HDF5 1.13.1. [Online]. Available: https://www.hdfgroup.org/2022/03/parallel-compression-improvements-in-hdf5-1-13-1/

[28] H. Tang, Q. Koziol, J. Ravi, and S. Byna, "Transparent Asynchronous Parallel I/O Using Background Threads," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 891–902, 2022.

[29] S. Jin, S. Di, J. Tian, S. Byna, D. Tao, and F. Cappello, "Improving prediction-based lossy compression dramatically via ratio-quality modeling," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 2494–2507.

[30] J. Li, W.-k. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel netcdf: A high-performance scientific i/o interface," in *SC'03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*. IEEE, 2003, pp. 39–39.

[31] W. F. Godoy, N. Podhorszki, R. Wang, C. Atkins, G. Eisenhauer, J. Gu, P. Davis, J. Choi, K. Germaschewski, K. Huck *et al.*, "Adios 2: The adaptable input output system. a framework for high-performance data management," *SoftwareX*, vol. 12, p. 100561, 2020.

[32] H. Tang, S. Byna, N. A. Petersson, and D. McCallen, "Tuning parallel data compression and i/o for large-scale earthquake simulation," in *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 2021, pp. 2992–2997.

[33] S. Byna, M. Chaarawi, Q. Koziol, J. Mainzer, and F. Willmore, "Tuning hdf5 subfiling performance on parallel file systems," *Lawrence Berkeley National Laboratory*, 2021.

[34] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[35] S. Jin, J. Pulido, P. Grosset, J. Tian, D. Tao, and J. Ahrens, "Adaptive configuration of in situ lossy compression for cosmology simulations via fine-grained rate-quality modeling," in *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, 2020, pp. 45–56.

[36] X. Liang, K. Zhao, S. Di, S. Li, R. Underwood, A. M. Gok, J. Tian, J. Deng, J. C. Calhoun, D. Tao, Z. Chen, and F. Cappello, "Sz3: A modular framework for composing prediction-based error-bounded lossy compressors," *IEEE Transactions on Big Data*, pp. 1–14, 2022.

[37] D. Tao, S. Di, Z. Chen, and F. Cappello, "Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization," in *2017 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2017, pp. 1129–1139.

[38] S. Di and F. Cappello, "Fast error-bounded lossy hpc data compression with SZ," in *2016 IEEE International Parallel and Distributed Processing Symposium*. Chicago, IL, USA: IEEE, 2016, pp. 730–739.

[39] Q. Zhou, C. Chu, N. Kumar, P. Kousha, S. Ghazimirsaeed, H. Subramoni, and D. Panda, "Designing high-performance mpi libraries with on-the-fly compression for modern gpu clusters," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2021, pp. 444–453.

[40] K. J. Bowers, B. Albright, L. Yin, B. Bergen, and T. Kwan, "Ultra-high performance three-dimensional electromagnetic relativistic kinetic plasma simulation," *Physics of Plasmas*, vol. 15, no. 5, p. 055703, 2008.

[41] S. Byna, J. Chou, O. Rubel, H. Karimabadi, W. S. Daughter, V. Roytershteyn, E. W. Bethel, M. Howison, K.-J. Hsu, K.-W. Lin *et al.*, "Parallel i/o, analysis, and visualization of a trillion particle simulation," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–12.

[42] B. Behzad, H. V. T. Luu, J. Huchette, S. Byna, R. Aydt, Q. Koziol, M. Snir *et al.*, "Taming parallel i/o complexity with auto-tuning," in *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2013, pp. 1–12.

[43] B. Behzad, S. Byna, S. M. Wild, M. Prabhat, and M. Snir, "Improving parallel i/o autotuning with performance modeling," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, 2014, pp. 253–256.

[44] W. Bhimji, D. Bard, M. Romanus, D. Paul, A. Ovsyannikov, B. Friesen, M. Bryson, J. Correa, G. K. Lockwood, V. Tsulaia *et al.*, "Accelerating science with the nersc burst buffer early user program," *Lawrence Berkeley National Laboratory*, 2016.

[45] J. Han, D. Koo, G. K. Lockwood, J. Lee, H. Eom, and S. Hwang, "Accelerating a burst buffer via user-level i/o isolation," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2017, pp. 245–255.

[46] D. Koo, J. Lee, J. Liu, E.-K. Byun, J.-H. Kwak, G. K. Lockwood, S. Hwang, K. Antypas, K. Wu, and H. Eom, "An empirical study of i/o separation for burst buffers in hpc systems," *Journal of Parallel and Distributed Computing*, vol. 148, pp. 96–108, 2021.

[47] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, V. Venkatram, L. Zarija, S. Saba, and W.-k. Liao, "Hacc: Simulating sky surveys on state-of-the-art supercomputing architectures," *New Astronomy*, vol. 42, pp. 49–65, 2016.

[48] Summit supercomputer. [Online]. Available: https://www.olcf.ornl.gov/summit/

[49] W. Daughton, J. Scudder, and H. Karimabadi, "Fully kinetic simulations of undriven magnetic reconnection with open boundary conditions," *Physics of Plasmas*, vol. 13, no. 7, p. 072101, 2006.

[50] A. K. Paul, O. Faaland, A. Moody, E. Gonsiorowski, K. Mohror, and A. R. Butt, "Understanding hpc application i/o behavior using system level statistics," in *2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE, 2020, pp. 202–211.

[51] S. Lee, K.-y. Hou, K. Wang, S. Sehrish, M. Paterno, J. Kowalkowski, Q. Koziol, R. B. Ross, A. Agrawal, A. Choudhary *et al.*, "A case study on parallel hdf5 dataset concatenation for high energy physics data analysis," *Parallel Computing*, vol. 110, p. 102877, 2022.

[52] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock, "I/o performance challenges at leadership scale," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. IEEE, 2009, pp. 1–12.

[53] B. Xie, H. Tang, S. Byna, Q. Koziol, and O. Sarp. Tuning I/O Performance on Summit – HDF5 Write Use Case Study. [Online]. Available: https://hps.vi4io.org/_media/events/2020/hpciodc20-hdf5.pdf

[54] S. Jin, G. Li, S. L. Song, and D. Tao, "A novel memory-efficient deep learning training framework via error-bounded lossy compression," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2021, pp. 485–487.

[55] S. Jin, C. Zhang, X. Jiang, Y. Feng, H. Guan, G. Li, S. L. Song, and D. Tao, "Comet: a novel memory-efficient deep learning training framework by using error-bounded lossy compression," *Proceedings of the VLDB Endowment*, vol. 15, no. 4, pp. 886–899, 2021.

[56] S. Byna, A. Uselton, D. K. Prabhat, and Y. He, "Trillion particles, 120,000 cores, and 350 tbs: Lessons learned from a hero i/o run on hopper," in *Cray user group meeting*, 2013.

[57] Bebop cluster. [Online]. Available: https://www.lcrc.anl.gov/systems/resources/bebop/

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

The paper reports the overall performance improvement of our proposed solution HDF5-SZ (i.e., write compressed data and overlap compression with I/O + compression schedule optimization) over two existing solutions (i.e., write original data, write compressed data with SZ lossy compression filter).

We ran the experiments on (1) the Summit supercomputer at Oak Ridge National Laboratory, each node of which is equipped with two IBM POWER9 processors with 42 physical cores and 512 GB DDR4 memory, and (2) the Bebop cluster at Argonne National Laboratory, each node of which is equipped with two 18-core Intel Xeon E5-2695v4 CPUs and 128 GB DDR4 memory. We compiled the HDF5-SZ demo code with GCC-8.3.1 and OpenMPI 4.1.1 with HDF5-1.10.6, Argobots-1.1, and SZ-2.1.

AUTHOR-CREATED OR MODIFIED ARTIFACTS:

Artifact 1

Persistent ID: `https://doi.org/10.5281/zenodo.6875597`
Artifact name: HDF5-SZ

Reproduction of the artifact without container: Please see the instructions in the Zenodo (https://doi.org/10.5281/zenodo.6875597) or GitHub repository (https://github.com/jinsian/HDF5-SZ) on how to compile and execute our demo code. We also describe the instructions as follows.

0 EXPERIMENTAL ENVIRONMENT

(1) OS: CentOS (>=7.8)
(2) Compiler: GCC (>=4.8.5)
(3) MPI: GCC built OpenMPI (>=4.1.1) or MPICH (>=3.3.1)
  (a) For users in HPC systems (such as Summit) with Slurm, please try "module load openmpi" to load the MPI library.
  (b) For users in Chameleon Cloud, please request a node in the user dashboard, create an instance using *ANL-MPICH* image, and launch and login to the instance.
(4) Other dependencies: parallel HDF5, Argobots, and SZ (please follow Step 1 and 2 to build them).

1 STEP 1: DOWNLOAD DEPENDENCIES

1.1 Setup test directory

```
export TEST_HOME=$(pwd)
```

1.2 Download code of HDF5, Argobots, SZ, and our HDF5-SZ

```
cd $TEST_HOME
git clone https://github.com/HDFGroup/hdf5
git clone https://github.com/pmodels/argobots
git clone https://github.com/szcompressor/SZ
git clone https://github.com/jinsian/HDF5-SZ
```

1.3 Configure home directory of each software

```
export H5_HOME=$TEST_HOME/hdf5
export ABT_HOME=$TEST_HOME/argobots
export SZ_HOME=$TEST_HOME/SZ
export VOL_HOME=$TEST_HOME/HDF5-SZ
```

2 STEP 2: BUILD DEPENDENCIES

2.1 Build parallel HDF5

```
# create makefile and installation dir
cd $H5_HOME && ./autogen.sh && mkdir install

# build and install parallel HDF5
./configure --prefix=$H5_HOME/install \
    --enable-parallel \
    --enable-threadsafe \
    --enable-unsupported
make -j8 && make install

# check if the installed HDF5 supports parallel mode
$H5_HOME/install/bin/h5pcc -showconfig
```

2.2 Build Argobots

```
# create makefile and installation dir
cd $ABT_HOME && ./autogen.sh && mkdir install

# build and install Argobots
./configure --prefix=$ABT_HOME/install
make -j8 && make install
```

2.3 Build SZ

```
# create makefile and installation directory
cd $SZ_HOME && mkdir install

# build and install SZ
./configure --prefix=$SZ_HOME/install
make -j8 && make install
```

2.4 Build asynchronous VOL connector with SZ

```
cd $VOL_HOME/src
export HDF5_DIR=$H5_HOME/install
export ABT_DIR=$ABT_HOME/install
make
```

3 STEP 3: BUILD HDF5-SZ DEMO

3.1 Set environment variables

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SZ_HOME/ \
install/lib
export LD_LIBRARY_PATH=$VOL_HOME/src:$H5_HOME/ \
install/lib:\
$ABT_HOME/install/lib:$LD_LIBRARY_PATH
```

```
export HDF5_PLUGIN_PATH="$VOL_HOME/src"
export HDF5_VOL_CONNECTOR="async \
under_vol=0;under_info={}"
```

3.2 Compile HDF5-SZ demo code

```
cd $VOL_HOME/test
export H5_DIR=$HDF5_DIR
export ASYNC_DIR=$VOL_HOME/src
make
```

4 STEP 4: TEST HDF5-SZ DEMO

4.1 Download dataset

You can download and extract the example dataset (i.e., a 2.7 GB Nyx cosmology dataset with a dimension of 512x512x512) from SDRBench with the following commands.

```
cd $VOL_HOME/test

# download dataset
wget https://g-8d6b0.fd635.8443.data.globus.org\
/ds131.2/Data-Reduction-Repo/raw-data/EXASKY/NYX/\
SDRBENCH-EXASKY-NYX-512x512x512.tar.gz

# extract data
tar -zxvf SDRBENCH-EXASKY-NYX-512x512x512.tar.gz
```

4.2 Run test

Run the overall performance test. Note that you may need to change the execution command from mpirun to the corresponding MPI launch command in your system. If you are using the MPICH on Chameleon, please change *mpirun* to *mpiexec*.

```
mpirun -n 16 overall_test.exe
```

This demo code is to first distribute the example data to 16 processors where each holds a partition of 6 64x64x64 data fields and then write these data to a shared HDF5 file using three different solutions (with different write performances): (1) write original data, (2) write compressed data with SZ lossy compression filter, and (3) write compressed data and overlap compression with I/O + compression schedule optimization.

You'll expect to see the output like this:

Write time comparison:
Original: 0.216752 s
Previous: 0.160252 s
Ours: 0.135717 s