Improving Performance of Data Dumping with Lossy Compression for Scientific Simulation

Xin Liang,* Sheng Di,† Dingwen Tao,‡ Sihuan Li,* Bogdan Nicolae,† Zizhong Chen,* and Franck Cappello†§
* University of California, Riverside, CA, USA
† Argonne National Laboratory, Lemont, IL, USA
‡ The University of Alabama, AL, USA
§ University of Illinois at Urbana-Champaign, IL, USA
xlian007@ucr.edu, sdi1@anl.gov, tao@cs.ua.edu, sli049@ucr.edu
bnicolae@anl.gov, chen@cs.ucr.edu, cappello@mcs.anl.gov

***Abstract*—Because of the ever-increasing data being produced by today's high performance computing (HPC) scientific simulations, I/O performance is becoming a significant bottleneck for their executions. An efficient error-controlled lossy compressor is a promising solution to significantly reduce data writing time for scientific simulations running on supercomputers. In this paper, we explore how to optimize the data dumping performance for scientific simulation by leveraging error-bounded lossy compression techniques. The contributions of the paper are threefold. (1) We propose a novel I/O performance profiling model that can effectively represent the I/O performance with different execution scales and data sizes, and optimize the estimation accuracy of data dumping performance using least square method. (2) We develop an adaptive lossy compression framework that can select the bestfit compressor (between two leading lossy compressors SZ and ZFP) with optimized parameter settings with respect to overall data dumping performance. (3) We evaluate our adaptive lossy compression framework with up to 32k cores on a supercomputer facilitated with fast I/O systems and using real-world scientific simulation datasets. Experiments show that our solution can mostly always lead the data dumping performance to the optimal level with very accurate selection of the bestfit lossy compressor and settings. The data dumping performance can be improved by up to 27% at different scales.**

I. Introduction

Today's high-performance computing (HPC) applications are producing massive amounts of data during simulations, which easily saturates the limited I/O bandwidth of the parallel file systems (PFS) typically used by supercomputing facilities for storage. For example, the Hardware/Hybrid Accelerated Cosmology Code (HACC) [1], operates with 1∼10 trillion particles in a single simulation [2]. This produces up to 220 TB of data for each particle snapshot, which quickly adds up to 22 PB of data for 100 snapshots taken over the entire simulation. Even considering a sustained bandwidth of 500 GB/s, the I/O time still exceeds 10 hours, making a full snapshot dump prohibitive. To address this issue, decimation is a commonly adopted approach: instead of storing every snapshot, the frequency is greatly reduced by storing only one out of every several snapshots. However, such an approach has a major drawback: it degrades the temporal constructiveness of the simulation and also loses valuable information that is needed for post-analysis. A better solution is to keep all snapshots but reduce their size. In this context, error-controlled lossy compression techniques have been shown to be capable of dramatically reducing the size (much more than lossless compression techniques) without significant degradation of the quality of the data.

Various lossy compressors have been proposed to address this issue in recent years. For instance, SSEM [3] is a wavelet-based lossy compressor used to improve the checkpoint time for large-scale climate simulations. ZFP [4] leverages in-orthogonal/orthogonal transforms to compress the simulation data for visualization purpose. SZ [7]–[11] has been proposed for both snapshot-based compression and time-based compression in but not limited to cosmological simulations such as HACC and NYX. Although these lossy compressors are all designed to significantly reduce the storage requirement and/or I/O performance, they have different characteristics by nature. For example, ZFP aims at providing fast in-situ compression thus it is usually very fast. On the other hand, SZ focuses more on compression ratio therefore it may be slower than ZFP but could offer higher reduction on size, especially for unstructured, 1D and 2D datasets. The naturally different design principles pose a challenging problem to the application users. How to select the best-fit compressor according to the application's needs is still an open question.

This paper focuses on the problem of how to improve the overall I/O performance of lossy compression used in the scientific simulations. The goal is to provide users with an error-bounded lossy compressor of possibly optimal output throughput. The key challenge that we address is how to optimize the trade-off between compression overhead, compression ratio when the quality of the data is guaranteed. This trade-off poses difficult challenges because of the antagonistic effects of these aspects: better compression ratio typically leads to higher compression overhead. Therefore, in order to optimize the I/O performance, we need to find the sweet spot that can minimize the sum between the compression overhead and the write overhead to the PFS without violating the desired quality properties. While compression can be applied in an embarrassingly parallel fashion, writes to the PFS are subject

978-1-7281-4734-5/19/$31.00 2019 IEEE

to a limited global I/O bandwidth. Therefore, it is non-trivial to find such a sweet spot. To solve this problem, we introduce an novel adaptive approach that relies on I/O performance modeling to characterize the write overhead to the PFS under variable number of ranks and data size per rank. Then, using this performance model and data sampling, we propose a strategy to choose an optimal lossy compression algorithm and fine-tune it such that it has the lowest dumping time.

Our contributions are summarized as follows:

- We propose an optimized data dumping performance model that can effectively represent the writing performance with different execution scales and data sizes.
- We analyze our proposed performance model in terms of the lossy compression technique, which is a fundamental guideline to develop an efficient algorithm for optimizing the data dumping performance.
- We develop an adaptive lossy compression framework with a series of optimization strategies to improve the dumping performance for the scientific simulations with error-bounded lossy compressors. The optimized framework has two critical steps: compression quality estimation and online optimization of compression settings.
- We conduct parallel experiments by processing two well-known scientific simulations (with a total of up to 100TB data) on ANL Theta supercomputer [12] with up to 32k cores. Experiments show that our optimized framework can improve the overall dumping performance by up to 27% when running with 32k cores.

The rest of the paper is organized as follows. In Section II, we discuss related work. In Section III, we describe the supercomputing platform to be used in our study and propose an effective profiling model for dumping performance. In Section IV, we analyze the dumping performance considering lossy compressors on a PFS. In Section V, we present our adaptive lossy compression framework with a series of optimization strategies for optimizing the overall dumping performance. In Section VI, we present the evaluation results from using three real-world simulation datasets on a supercomputer. In Section VII, we present our conclusion and discuss our future work.

II. Related Works

A. Lossy data compression for scientific datasets

Scientific data compression has been studied for decades. The data compressors can be categorized into two categories: lossless and lossy. Lossless compression [13]–[15] suffers from very low compression ratio on scientific data, because the binary representations of floating-point values may hardly contain identical symbols or exactly duplicated chunks. Error-bounded lossy compressors [4]–[11] are proposed under this circumstance to trade certain accuracy for higher compression ratio. They guarantee the distorted data could be used in user's post-analysis due to the error-bounded feature, thus can be applied in many situations [16].

Among all the possible objectives for error-bounded lossy compression, dumping performance is one of the most important when the fidelity of the decompressed data is guaranteed, as the compression is usually done in-situ with the scientific applications. ZFP is designed to be as fast as possible so that the overhead on compression could be minimized. However, as the application also needs to dump the compressed data into file system, the dumping time could be another concern as I/O bandwidth is usually limited, especially when the scale is large. SZ alleviates this problem by designing a slower compressor with more analysis on the data, which could offer higher compression ratios at most times. But it suffers from the high compression time when the dumping performance is high or total data size is small.

Previous work [17] investigates the scalable performance of various compression algorithms and analyzes the I/O performance regarding compression. Unlike our work targets optimizing the data dumping performance, it focuses only on how to evaluate compressors using synthetic data as comprehensively as possible.

B. I/O modeling of parallel file systems for lossy compression

A large amount of work has been dedicated to modeling and prediction of I/O access patterns, which is then used to optimize various storage aspect. Approaches such as [18], [19] use auto-encoders and machine learning to understand variability of I/O operations. Approaches such as [20], [21] use auto-regressive moving average (ARIMA) to perform online predictions for the purpose of pre-fetching. In the context of cloud computing, an important problem is the study of I/O interference due to multi-tenancy. To this end, approaches such as [22] model I/O performance using Petri nets. Another related cloud problem is how to deploy a distributed file system in a cost-effective manner. To this end, [23] proposes a modeling approach that specifically targets practical performance analysis. In the context of HPC, several efforts aim to understand and characterize the I/O behavior of parallel file systems by analyzing the historical utilization [24], [25].

While many of the aforementioned approaches provide in-depth insight into utilization patterns of parallel file systems and can be used to predict the I/O performance, they do not consider the impact of lossy compression, which is a key differentiator in our proposal. To the best of our knowledge, we are the first to study this aspect. Our performance model builds a roof-line model via least-square by considering file size per rank. Many related approaches are complementary to this model and can be used to further enhance our proposal.

III. I/O Performance Model of Parallel File Systems under Write Concurrency

In this section, we introduce an I/O performance model for parallel file systems to address the concurrent data dumping scenario we explore in this paper. Specifically, lossy compression usually leads to similar compression ratios for similar data, which means all application processes write roughly the same amount of data concurrently. Therefore, the question our performance model aims to answer is what I/O bandwidth can be achieved if N application processes (ranks) need to write M GB of data simultaneously to the parallel file system?

To answer this question, we introduce a methodology to build a performance model that we illustrate using the Theta [12] supercomputer, ranked 24th as of November 2018 in the Top 500 [26] and featuring a 10 PB Lustre parallel file system. The same methodology can be applied for other supercomputers and parallel file system configurations.

As a first step, we analyze the write throughput on Theta under weak scalability (increasing number of writers) and different data size per writer. We assume the application consists of N ranks, each of which needs to dump M GB of data per snapshot. Although the parallel file system has a theoretical peak I/O bandwidth P, it is not enough to simply approximate the total write time as $M \cdot N/P$. This is because the degree of write concurrency and the size per writer can lead significant variability even when the total size $M \cdot N$ remains the same. Furthermore, the behavior of the PFS can become even more complex when its peak bandwidth is not saturated. To account for these factors, we consider the overall write bandwidth as a function of the file size per node (x) and the number of ranks (N): $f(x, N)$ GB/s. In this case, the time dump the whole snapshot onto the PFS can be approximated as t_w=$\frac{MN}{f(M,N)}$.

To obtain f, we perform an I/O performance profiling with different execution scales (i.e., we consider a set of usual values for N). Through experimentation, we observed that f tends to grow linearly with the logarithm of M up to a threshold (below the peak bandwidth P), after which it remains constant. These experimental results are illustrated in Figure 1. Such an observation is consistent with the intuitive behavior of parallel file systems, as the peak bandwidth is hardly achievable given overheads and limits that could be set by the system administrator. Therefore, we propose to formulate f using a roof-line model, as shown in Formula (1).

$$f(x) = \begin{cases} ax + b & x < x_t \\ c & x \geq x_t \end{cases} \tag{1}$$

where x refers to the logarithm of the data size per rank and x_t is the turning point indicating the saturation of the I/O bandwidth. To obtain the coefficients a, b, c, we use least-square fitting of the experimental data (which uses a variable size from 2^1 MB to 2^9 MB and covers the typical file sizes generated by lossy compression). Then, once the coefficients were determined, we can use f to predict the performance for any data size.

Unfortunately, the point of saturation x_t is not the same for all N. Therefore, we assume that x_t is a function of N, which leads to a separate optimization problem for all usual values of N we consider (denoted by i). The process of determining the parameters in the roof-line model can be formulated as Equation (2).

$$f_i(x) = \begin{cases} ax + b & x < x_i \\ c & x \geq x_i \end{cases} \tag{2}$$

where $\{x_i\}$ and $\{y_i\}$ are referred to the turning point (i.e., the profiling case i) and the corresponding I/O bandwidth, respectively. Specifically, for each execution scale with various data sizes per rank, the least-square optimization problem can be formulated as follows.

We need to identify the coefficients a, b and c that can minimize $\sum_j (f_i(x_j) - y_j)^2$, where i is assumed to be the profiling case number (as shown in Formula (3)).

$$\min_{a,b,c} \sum\nolimits_j (f_i(x_j) - y_j)^2 \tag{3}$$

We denote the corresponding least-square optimized coefficients as a_i^*, b_i^*, and c_i^*, respectively. Then, the optimal profiling case number (denoted by i^*) can be determined by the following equation (i.e., the one leading to the least-square).

$$i^* = \arg\min_i \{ \min_{a_i^*, b_i^*, c_i^*} \sum\nolimits_j (f_i(x_j) - y_j)^2 \} \tag{4}$$

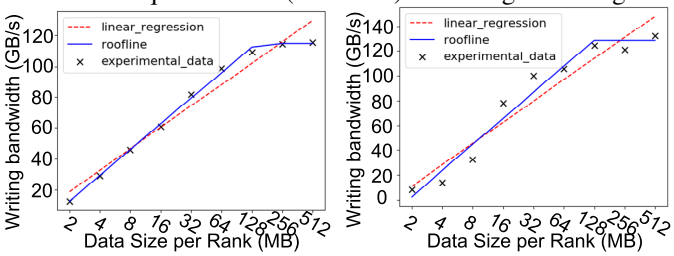
Then, the optimal turning point (denoted by x_t^*) can be calculated as follows:

$$x_t^* = \frac{c^* - b^*}{a^*} \tag{5}$$

where a^*, b^* and c^* refer to the corresponding coefficients with the turning point being set to the profiling case i^*. The final optimal I/O model is presented in Formula (6)

$$f^*(x) = \begin{cases} a^*x + b^* & x < x_t^* \\ c^* & x \geq x_t^* \end{cases} \tag{6}$$

There is a specific situation we have to deal particularly with during the above model optimization. In fact, the above calculation might result in a situation that c^* is smaller than $a^*x_t^* + b^*$, implying that the I/O saturated case has lower bandwidth than does the I/O unsaturated simulation with unsaturated I/O, which is a little counter-intuitive. Hence, we need to do an adjustment to our I/O model to respect the ground-truth. If this counter-intuitive situation occurs, we set c to $a^*x_t^* + b^*$ in the roof-line model and recompute the error and corresponding parameters. This can always give us a valid roof-line model, where the constant part is always no less than the maximal of the linear part. Fig. 1 shows the experiments and the resulting roof-line model for the data writing bandwidth on theta with 256, and 512 nodes. The roof-model (blue curve) does provide a better trend compared to vanilla least-square model (red curve) according to this figure.



(a) Write BW (256 nodes) (b) Write BW (512 nodes)

Fig. 1. Write/Read bandwidth (BW) on Theta

Note that the above-mentioned I/O performance profiling just needs to be done once for each supercomputer and our optimized I/O model based on the profiling results can be reused for all the applications running on the supercomputer. Thus, the I/O performance profiling is not overhead that needs to be paid every time an application is running.

IV. IO Model with Compression

Our target is to optimize the data dumping performance by lossy compression such that the simulation data can be dumped

to a PFS as fast as possible, while still guaranteeing that the data is valid for user's post-analysis. This contrasts other research with multi-objectives regarding lossy compression [4], [6], [27].

Adding compression to the I/O performance model makes the problem a trade-off, since the compression introduces extra time on compression while saving cerntain time on data writing because of reduced data size. In the following text, we extend the I/O model by taking the impact of error-bounded lossy compression into consideration.

The error-bounded lossy compressor allows users to compress the data based on a specific error setting (absolute error bound, relative error bound, or PSNR), and its impact on the original I/O model can be formulated by some functions of the given error setting. In our analysis, we focus on absolute error bound (denoted as eb) because it has been widely used in the community. In general, compression ratio, compression rate and decompression rate are determined by the absolute error bound. Suppose the lossy compressor achieves a compression ratio[1] of $\rho(eb) : 1$ and the minimum compression rate is denoted as $f_{comp}(eb)$ GB/s. Since each rank compresses its local data independently, there is no communication cost across ranks (except the negligible Allreduce operation for synchronization which determines the final size), thus the overall data dumping time can be formulated as $t_{dump}(eb) = \frac{MN/\rho(eb)}{f_{write}(M/\rho(eb),N)} + \frac{M}{f_{comp}(eb)}$. The overall data dumping rate with respect to the given error bound *eb* can be derived as Formula (7).

$$Rate_{dump}(eb) = (\frac{MN/\rho(eb)}{f_{write}(M/\rho(eb),N)} + \frac{M}{f_{comp}(eb)})^{-1} \quad (7)$$

Based on the above formula, we note that the key factors affecting the compression-based data dumping rate include data writing rate function f_{write}, compression ratio ρ, and compression rate f_{comp}. Even though the execution scale is fixed, there is still a complicated interplay among them. Although the intermediate cases are complicated, we can interpret some extreme cases intuitively. Considering each rank handles the same amount of simulation data, f_{write} would be much larger than f_{comp} if the simulation runs in a relatively small scale, such that a faster compressor would be more favored. On the other hand, if the execution scale is extremely large, f_{write} could be close to the peak performance of the file system (a situation with saturated I/O). In this situation, larger N would increase writing time linearly, in which a compressor featuring higher compression ratio should be considered. Therefore, having a good knowledge of the I/O information is essential to designing an adaptive compressor with the best I/O performance.

In fact, our model is not restricted to only the I/O use-case. It can be extended to any other scenarios associated with compression involving a resource having a performance that can be modeled as a roof-line. For instance, if one wants to leverage compression to accelerate the data transfer between memory and cache, the time t_{dump} could be replaced by the data transferring time. Similarly, if compression is used in data communication, t_{dump} corresponds to the communication time for the message.

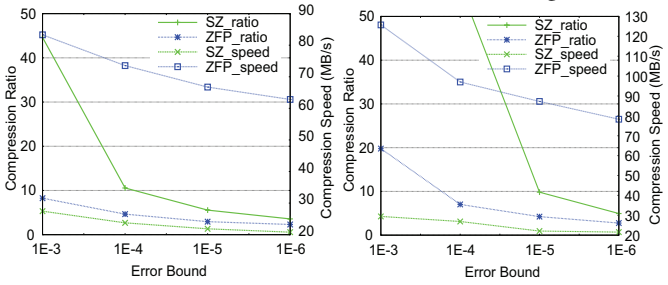
[1]Compression ratio is the ratio of original data size to compressed size.

V. Adaptive Lossy Compression Framework

In this section, we present the design of our adaptive lossy compression framework for getting the best overall parallel data dumping performance. In the following text, we first introduce the design motivation, and then present our solution in details. Because of limited space, we only show the examples done on the Theta supercomputer in this section as it represents the modern supercomputers with fast I/O.

A. Design Motivation

Our target is to design an adaptive framework that can maximize the data dumping performance by automatically selecting the bestfit compressor, given the basic profiling information of the I/O environment (such as sampled I/O bandwidth based on execution scale and data sizes). In our framework, we choose two state-of-the-art lossy compressors - SZ and ZFP, because they have been considered the best two error-bounded lossy compressors on different datasets in literature [4], [9]. In general, SZ has higher compression ratios for fixed error bounds while ZFP has higher compression rate, especially because of their different compression principles. SZ contains four critical steps: (1) predict each data value, (2) perform linear-scaling quantization, (3) a customized variable-length encoding, and (4) optional lossless compression such as Zstd [28]. By comparison, ZFP contains five critical steps: (1) align values in each block to a common exponent; (2) convert floating-point values to a fixed-point representation; (3)decorrelation by applying orthogonal transforms; (4) order transform coefficients; and (5) embedded encoding.



(a) velocity_x (b) dark_matter_density

Fig. 2. Performance of different compressors on two typical fields in NYX dataset on Theta

Figure 2 shows the compression ratio and compression speed for two example fields in the NYX dataset. From this figure, we can see that SZ leads to much higher compression ratios given the error bound, especially on some easy-to-compress dataset such as dark_matter_density. The compression ratio could be 10X higher than that of ZFP. However, this does not imply it will always lead to better overall data dumping performance. This is because the compression speed of SZ is only one fourth of that of ZFP, such that the performance gain achieved from data reduction cannot beat over the compression time cost especially when the the data size is

small (e.g. in small execution scale or with large compression ratios). As such, selecting the bestfit compressor is critical to the overall data dumping performance. Furthermore, we also perform some optimization steps and fine-tuning on the compressors to achieve higher data dumping performance.

B. Design Overview

We present the design overview of our adaptive lossy compression framework in Figure 3. It has a sampling stage at the very beginning to collect the information for the different compressors. After that, we have a decision process to determine which compressor to use and whether lossless compression should be performed on the compressed data. If the selected compressor is SZ, we perform further optimization regarding the tradeoff between compression ratio and compression speed. Specifically, we re-design the key stages in the SZ compression and offer more options to fine-tune the compressor. We also perform a fast parameter search to get the best parameter settings. We then perform the compression using the selected bestfit compression strategy (either ZFP or fine-tuned SZ). Finally, we get the maximum size of the compressed data and pack all the data segments to the multiple of 64KB to maximize the I/O bandwidth, as required by the Lustre file system on Theta (stripe size) for the best performance, and write the files into the parallel file system.

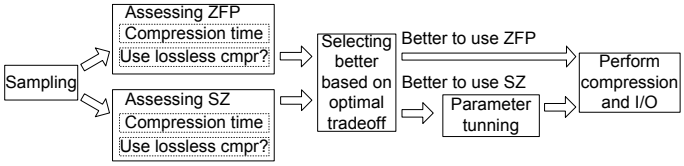


Fig. 3. Design Overview of Adaptive Lossy Compression Framework

TABLE I
VARIABLE DEFINITIONS

N	Number of ranks used
M	File size on each rank
t_{comp}	Lossy compression time
t_{ll_comp}	Lossless compression time
ρ_l	Lossy compression ratio
ρ_{ll}	Lossless compression ratio
$f_{write}(x)$	Write bandwidth collected from previous section

We list the key notations to be used in Table I. Based on the definition of these notations, we can infer the total dumping performance with/without the compressor as well as with/without lossy compression by Equation (7) as follows:

$$Rate_{dump} = \frac{f_{write}(M)}{MN} \tag{8}$$

$$Rate_{dump\_l} = \frac{1}{\frac{MN/\rho_l}{f_{write}(M/\rho_l)} + t_{comp}} \tag{9}$$

$$Rate_{dump\_ll} = \frac{1}{\frac{MN/\rho_{ll}}{f_{write}(M/\rho_{ll})} + t_{comp} + t_{ll\_comp}} \tag{10}$$

C. Block-wise data sampling and statistics estimation

According to the analysis in the previous section, we need to know the compression ratio and compression speed to compute the corresponding dumping performance given the I/O information and error settings. However, it is very hard to find some surrogate function to do so without data information since the compression ratio as well as the compression speed may depend on specific datasets and/or different error settings significantly. Some of existing works [29] adopted sampling methods to estimate the compression ratio roughly for these compressors. However, they overestimated the compression ratios for SZ as they used the entropy formula which represents an ideal situation that is hardly reached by the compressor (even when SZ is equipped with outstanding lossless compressors such as Zstd) in practice. For the performance estimation of ZFP, the existing sampling methods either overlooked the meta data in the embedded encoding or did not study the impact of lossless compression. Furthermore, they did not have any study on the compression speed since they focus on on compression ratios.

In our approach, we extend the sampling idea to estimate both the compression ratios and compression speed for the two compressors. Apart from the compression ratio estimation, we notice that sampling can offer an effective compression speed approximation based on our in-depth analysis of the compression algorithms. ZFP, for example, performs a customized transform, fixed point alignment and embedded encoding individually on each 4x4x4 block. Its compression time is linear to the number of blocks as well as the time to compress each block. The sampling method would give us a quite accurate estimation in this case. As for SZ, the most time-consuming stage would be the block-wise prediction and quantization, together with the following Huffman encoding. The operations in the block-wise prediction and quantization would be the same for each data block, which also implies linearity in the compression time. As for the Huffman encoding, the sampling would also give good approximations as long as the sampled data leads to similar distributions of quantization index (which likely appears as long as the sampling is not biased).

The detailed sampling method we designed is described as follows. We need to sample the data in common multiples (12x12x12) of 6x6x6 and 4x4x4 because the default 6x6x6 block setting in SZ would usually lead to a better result. To restrict the overhead, we restrict the sampled data to be around 1% of the original data and this already leads to a great performance gain (to be shown later). Specifically, we use a uniform sampling method in the granularity of 12x12x12 block. That is, we sample one 12x12x12 block from every certain number of blocks along each dimension of the data. The distance is fixed for every dimension and is restricted by the 1% requirement. For example, we have 512x512x512 data in the NYX dataset, which corresponds to 42x42x42 blocks. Then, we slightly increase the sampling until around 1% sampling rate is reached. In this case, we will sample 1 every 4 blocks along each dimension, leading to around 1.7% sampling rate. This sampling approach would produce a deterministic result, which is also the reason why we did not adopt the randomize algorithms.

Table II shows the sampling accuracy of the statistics (listed in Table I) on two typical fields in the NYX dataset. Please

TABLE II
ACCURACY OF THE UNIFORM SAMPLING APPROACH

			t_{comp}			t_{ll_comp}			ρ_l			ρ_{ll}		
field	**eb**	**compressor**	**real**	**estimated**	**error**	**real**	**estimated**	**error**	**real**	**estimated**	**error**	**real**	**estimated**	**error**
velocity_x	1e-3	SZ	12.36	11.66	-5.7%	3.62	2.91	-19.6%	14.92	15.81	6.0%	19.33	21.60	11.7%
		ZFP	6.11	5.74	-6.2%	1.45	0.94	-35.2%	8.24	8.61	4.6%	8.52	8.89	4.4%
	1e-4	SZ	12.92	12.36	-4.3%	0.54	0.67	24.9%	11.29	11.44	1.3%	11.82	11.77	-0.4%
		ZFP	6.95	6.38	-8.1%	1.37	1.43	4.4%	4.65	4.87	4.7%	4.65	4.87	4.7%
	1e-5	SZ	14.10	13.69	-2.9%	0.53	0.75	41.4%	5.76	5.71	-0.9%	5.76	5.78	0.3%
		ZFP	7.66	6.95	-9.3%	2.14	1.98	-7.3%	2.94	3.08	4.7%	2.94	3.08	4.7%
	1e-6	SZ	15.53	17.28	11.2%	0.86	1.36	58.5%	3.59	3.41	-5.2%	3.60	3.56	-1.1%
		ZFP	8.14	7.63	-6.3%	2.70	2.71	0.6%	2.31	2.41	4.8%	2.30	2.41	4.8%
dark matter density	1e-3	SZ	6.07	5.76	-5.1%	0.15	0.22	46.2%	31.41	32.81	4.5%	578.95	557.89	-3.6%
		ZFP	4.05	3.87	-4.5%	2.79	2.44	-12.7%	19.80	20.64	4.2%	24.98	25.72	3.0%
	1e-4	SZ	7.17	6.83	-4.8%	1.36	1.17	-14.0%	15.06	15.75	4.5%	60.40	61.81	2.3%
		ZFP	5.26	4.88	-7.2%	7.83	6.88	-12.1%	7.03	7.36	4.7%	7.59	7.93	4.5%
	1e-5	SZ	12.44	12.11	-2.6%	2.98	0.83	-72.0%	9.27	9.55	3.0%	9.70	9.90	2.1%
		ZFP	5.86	5.31	-9.3%	11.46	10.45	-8.8%	4.24	4.44	4.7%	4.37	4.57	4.7%
	1e-6	SZ	14.87	15.26	2.6%	0.89	0.91	2.7%	4.91	4.83	-1.6%	4.92	4.97	1.0%
		ZFP	6.52	5.79	-11.2%	12.93	9.87	-23.7%	2.77	2.90	4.8%	2.77	2.90	4.8%

note that the SZ result here denotes the improved SZ with fastest setting (to be detailed later) as we need to compare this version with ZFP in our final solution. The column "real", "estimated" and "error" stand for the real execution time, estimated execution time from sampling, and the corresponding relative error of estimated time with respect to the real time, respectively. From this table, we can see that the uniform sampling approach already get a satisfactory accuracy as most of the error for t_{comp}, ρ_l and ρ_{ll} is within 5%. The only exception is t_{ll_comp}, which indicates the lossless compression time on the lossy compressed data. This is because the lossless compression time relies heavily on the data layout so it expresses less linearity on the data size. However, on the one hand, as lossless compression time only takes a small percentage compared to the SZ compression (around 5% on average), this inaccuracy would have little impact on the final result for SZ. On the other hand, although the lossless compression time may be higher than the compression time of ZFP (e.g. error bound $1E{-}5$ and $1E{-}6$ in dark_matter_density), our framework would make an easy decision to not apply the lossless compression due to the high time overhead and little benefit in the compression ratio in normal cases. All in all, this sampling approach would provide us descent results, as will be validated in the evaluation section.

D. Adjustments and Optimizations on SZ

We further improve the performance and flexibility of the SZ compressor by applying some adjustments and optimizations to either improve the performance or provide more chances to get better speed-ratio tradeoffs because of more options we create. This is inspired by the fact that the performance gain actually can be further improved if we choose SZ, compared with the best selection of original SZ and ZFP. From our experiments, we obverse that SZ usually "over-compress" the data with respect to the overall dumping performance. For example, SZ spends 20 seconds to compress the velocity_x field in the NYX dataset by 40X when the error bound is set to 1E-3, leading to less than 4 seconds in writing with 256 nodes on Theta due to the significant reduced data size thus 24s total dumping time. On the other hand, if we can use less computation to compress the data with a lower compression ratio, say 10 seconds to compress the data by 10X, we end up with around 9 seconds on data writing and thus 19 seconds on total data dumping time (because 40X compression corresponds to around 54GB/s writing performance while 10X compression corresponds to around 90GB/s in terms of Figure 1). In this case, we improve the overall performance of SZ by 26% due to 2X faster compression speed with 4X lower compression ratio. As an important contribution of this paper, we explore how to optimize SZ to get the best ratio-speed tradeoffs in different situations.

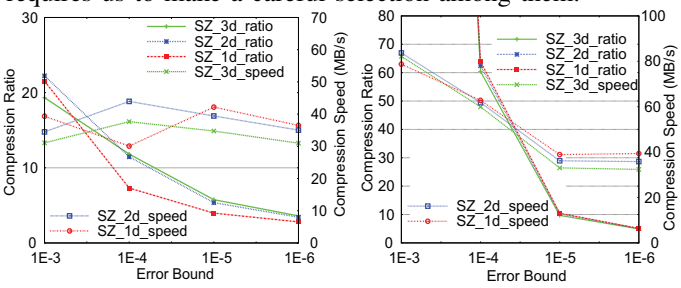
1) Online Determination on Lossless Compression: Although SZ provides a configuration file allowing users to decide whether to apply lossless compression, it cannot make an online decision automatically. By contrast, our adaptive framework is able to decide whether to use the lossless compression stage in SZ under different I/O environment, as presented in previous section. We note it here as it is a good example to offer the flexibility for the ratio-speed tradeoff.

2) Lower Precision Operations Alignment: In our previous SZ implementation, we adopt double precision for most of the computation as it gives more accurate result than the single-precision, avoiding the round-off errors effectively. Thus, when the data is in single precision, there would be many implicit type conversions between double and float, which may incur noticeable performance overhead. This is especially observed for the KNL nodes on Theta. Therefore, we align all the variables used in the SZ prediction and quantization to the data precision and eliminate the conversion overhead and accelerate the computation since single-precision operations are usually faster than the double precision operations, especially in the KNL nodes on Theta. Furthermore, this alignment would have little impact on the final compression ratio because single-precision is accurate enough to perform the error-bounded lossy compression for single-precision datasets.

3) Low Dimensional Prediction and Unpredictable Data Re-organization for Dependency Removal: We also re-design the prediction scheme in SZ to provide higher feasibility to tradeoff between compression ratios and compression speeds. Specifically, We study the ratio-speed tradeoffs for the prediction dimensions in SZ and remove the dependencies to let the compiler have a better optimization for the code.

We consider to revise the 3D prediction in the Lorenzo [30]

predictor for higher performance as the heavy dependencies in the Lorenzo predictor greatly prevents the compiler from optimizing the code. In the 3D Lorenzo predictor, each data point depends on all its 7 adjacent neighbors, making it impossible for parallelism. To alleviate this problem, we propose to use lower dimension prediction for less dependencies and less computations. For instance, if 2D prediction is used for 3D data, the dependency only exist along the 2D plane. Then, the operations along the other dimension could potentially be done using some instruction-level-parallelism techniques. Furthermore, if 1D prediction is used, the operations along the other two dimensions could both be done in parallel. Besides, SZ also have a special strategy for unpredictable data (which is far away from predicted value and have to be stored separately). We also isolate this process to remove the dependencies for the same purpose. Consequently, this adjustment could lead to worse compression ratio as higher dimensional prediction usually leads to better accuracy, which requires us to make a careful selection among them.



(a) velocity_x (b) dark_matter_density

Fig. 4. Performance of different predict dimensions on two typical fields in NYX dataset

The compression/decompression speed and the corresponding compression ratio for prediction using different dimensions are shown in Figure 4. From this figure, we can see that the 3D prediction can usually leads to the best compression, while it has the worse compression/decompression speed due to more computation operations and less potential for parallelization, which may not be the best option in some cases. While 1D prediction is unstable sometimes (probably due to the different number of unpredictable data), it always leads the compression speed when the error bound is relatively small. For instance, we would rather use 1d prediction for the dark_matter_density field, as it has 20% higher compression speed while keeping almost the same compression ratio.

E. Simplified parameter search

Besides the optimizations on performance, we do a simplified parameter search to the reach the best configuration for SZ as it is a highly tunable compressor with many parameters such as block sizes, prediction dimensions, max number of quantization index, etc. However, as each parameter combination requires a synchronization on all the ranks (otherwise we may have some ranks with high compression speed and low compress ratio while others with low compression speed and high compression ratio, which gives us the worst result), the number of searches should be minimal. After numerous analysis and experiments, we identify block size and prediction dimension as the most important parameters in SZ. We then use a heuristic to explore how to reach the best parameter setting on the two parameters with minimal overhead.

1) Block size and all Lorenzo prediction: The latest SZ 2.0 [9] adopts a block-wise design which automatically selects between a Lorenzo predictor and a linear-regression based predictor for each block. It uses a block size of 6 by default, as it seems to be the most reasonable option across different datasets and error settings [9]. However, it is never the optimal in any case. Generally speaking, a small block size (but not too small) is preferred when the error bound is loose (e.g. 1E-2) as it can predict data more accurately. However, as smaller block sizes introduce larger overhead, it will affect the performance when the error bound is tight (e.g. 1E-5). On the other hand, we find that the best setting of tight error bounds is to use Lorenzo prediction in all the blocks, as it eliminates all the necessary computation related to regression (which potentially increases the speed) and does not waste any space on storing the regression coefficients and the corresponding meta data (which potentially increases the ratio). Therefore, we select the better one between the default block size 6 and all Lorenzo prediction (in fact it is the extreme case when the block size is the data size and Lorenzo predictor is chosen for the only block), which can already lead to optimal solutions with relatively tight error bounds. For the loose error bound, we stick to the block size 6 as 1) we can reuse the sampled data; 2) it has similar performance to the optimal block size; and 3) the benefit from slighter higher compression ratio would not be much because the I/O time is not dominant in this case and the I/O bandwidth will drop.

2) Prediction dimension search: Another key parameter is prediction dimension discussed in the previous subsection, as it could lead to up to 20% difference in compression time and 50% compression ratios between 1D and 3D predictions as shown in Figure 4. Therefore, we do another parameter search along the prediction dimension to get the optimal one. As 1D prediction is the fastest, we will start there and check how it is compared to 2D prediction. If it is better, we will return 1D prediction as the optimal solution. Otherwise, we will continue to search between 2D prediction and 3D prediction.

Algorithm 1 concludes our parameter search approach. The *sync* operation in the pseudo code indicates the MPI_Allreduce operation to collect the best selection from all the ranks. It will gather the votes for the best selection and then collect the corresponding lossless settings. As we are given the profile for SZ with fastest setting (when compared with ZFP), we perform SZ with the regression setting (block size 6 with 3D prediction) on the sample data to gather the information. We then select the one with the lower estimated time as before. If the regression setting works better, we will use the default regression setting directly as further parameter adjustment would make little difference to the final result. On the other hand, if the all Lorenzo setting works better, it indicates either the error bound is relatively low (the compression ratio is relatively high) or the compression time

Algorithm 1 HEURISTIC TO EXPLORE BEST PARAMETER SETTING FOR SZ

Input: sampled data D, IO information io, estimated time t_0 for fastest setting, fastest setting $s0$
Output: Optimal SZ parameter setting

```
s_1 = SZ_WITH_REGRESSION_AND_1D_PRED;
t_1 = estimate_time(D, io, s_1); sync(); /*Estimate the time for the regression setting*/
if (t_0 < t_1) then
    s_1 = s_0;s_1.pred_dim = 2; /*Explore 2D prediction*/
    t_1 = estimate_time(D, io, s_1); sync();
    if (t_0 < t_1) then
        return s_0; /*Return setting with 1D prediction when 1D prediction is better than 2D prediction*/
    else
        s_0.pred_dim = 3; /*Explore 3D prediction*/
        t_0 = estimate_time(D, io, s_0); sync();
        if (t_0 < t_1) then
            return s_0; /*Return setting with 3D prediction*/
        else
            return s_1; /*Return setting with 2D prediction*/
        end if
    end if
else
    return s_1 /*Return the regression setting*/
end if
```

is more important. In this case, we will continue search along prediction dimensions to find the best one. Please note that we will stop exploration for 3D prediction when 1D prediction is better than 2D prediction as it is very unlikely to be better than 1D prediction in this case. This early stopping criterion will save one execution on the sample data.

VI. PERFORMANCE EVALUATION

In this section, we present the performance evaluation results on two supercomputers to show the efficiency of our adaptive compression framework.

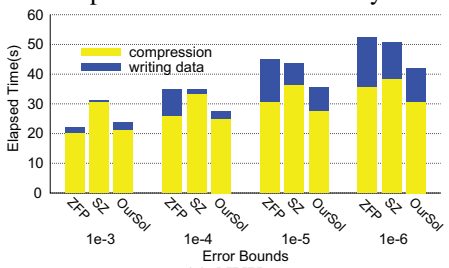
A. Experimental Setting

We conducted our experimental evaluation on two supercomputers with different scales. We first evaluate our framework on Bebop [31], a medium-scale supercomputer with relatively low bandwidth I/O facilities. Bebop has 1024 public nodes, each with two Intel Xeon E5-2695 v4 processors and 128GB of memory, and each processor with 18 cores. It uses General Parallel File Systems (GPFS) equipped with 2 I/O nodes, providing around 2GB/s I/O bandwidth. Bebop is a good example for extreme case when I/O is saturated all the time (which is usually the case for scientific simulations), as its 2 I/O nodes are always used. We also validate our framework on Theta [12], a supercomputer with relatively low CPU frequency (KNL nodes) and high I/O bandwidth. Each node on Theta is a 64-core processor with 16GB high bandwidth in-package memory and 192 GB of DDR4 RAM. It is also equipped with 10 PB Lustre file system [32] with a total bandwidth of 172 GB/s for write and 240 GB/s for read, which is representative of realistic systems with high-performance I/O. We use MPI-IO [33] for all the experiments as it is efficient and widely used. The application data is from 2 typical scientific simulations, namely NYX [34] cosmology simulation and SCALE-LETKF [35] weather simulation. Each application involves six data fields, each of which contains 512MB/540MB data on each rank for the two applications, respectively. This yields to a total of 96TB/101.25TB data on Theta when the scale is 512 nodes.

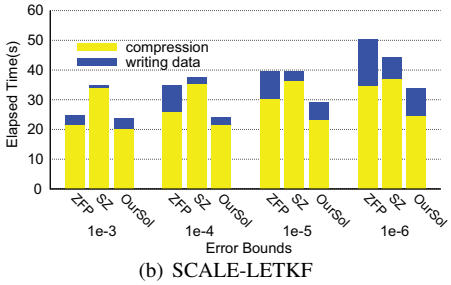
We focus on the overall dumping performance on the parallel file systems, including both compression time and writing time. We applied our solution to two state-of-the-art lossy compressors - ZFP [4] that often has higher compression speed and SZ [9] which generally leads to higher compression ratios. In the following sections, we first show the dumping performance of our adaptive framework compared to these two state-of-the-art compressors, and then we show a detailed breakdown of the time and overhead.

B. Dumping performance on Bebop

We first present the dumping performance on the Bebop cluster. Specifically, we first show the result on only 1 node. We choose this scale because it is the turning point on selecting between ZFP and SZ for most of the fields. It happens at such a low scale because 1) ZFP does not have as much dominance in speed as that on Theta; 2) I/O bandwidth in Bebop is very low such that compression ratio would easily dominate.



(a) NYX



(b) SCALE-LETKF

Fig. 5. Dumping performance of different error bounds (1E-3 $\sim$ 1E-6) on Bebop with 1 node (32 cores)

The overall dumping performance on Bebop with 1 node is shown in Figure 5. From this figure, we can see that the compression time takes the major part of the total dumping time. Even in this case, ZFP can only be better than SZ when the error bound is large because it only has slightly higher compression speed than that of SZ, which could not compensate for the loss in writing time on this platform. On the other hand, our solution always leads to the shortest dumping time except for the error bound 1E-3 on NYX dataset, because the adaptive framework is able to switch to the most suitable option for the different fields in the dataset. As for the error bound 1E-3 on NYX dataset, our solution selects ZFP correctly for all the fields, with a slight overhead because of the sampling and other decision process in our design.

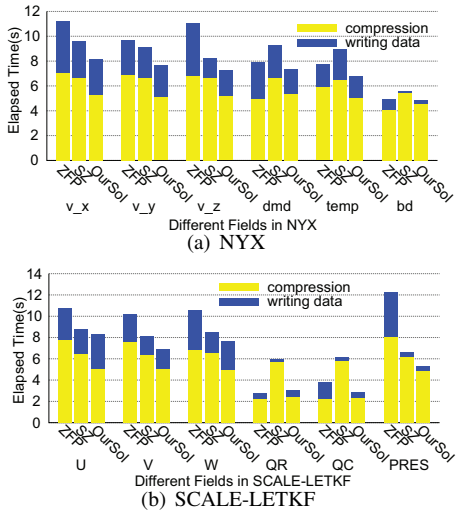


Fig. 6. Detailed breakdown of I/O Performance on each field (error bound 1E-6) on Bebop with 1 nodes (32 cores)

We also show the detailed time breakdown on the six fields of the two datasets in Figure 6, where we can easily figure out how our adaptive framework makes the decision. For instance, the adaptive framework selects SZ for all the fields, and it discards the lossless compression in SZ for the last 4 fields. Therefore, although our solution may have higher writing time, it has much lower compression time thanks to our optimization of parameter settings and the removal of lossless compression.

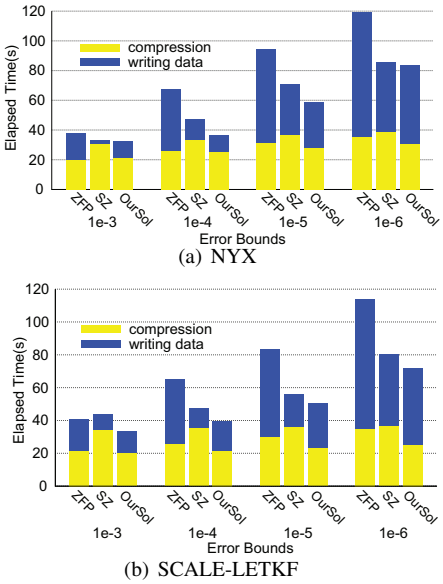


Fig. 7. Dumping performance of different error bounds (1E-3 $\sim$ 1E-6) on Bebop with 4 nodes (128 cores)

We then perform the experiments with 4 nodes on Bebop. In this case, our solution also leads to the best overall dumping performance. Please note that our solution costs more time than SZ in the ”bd” field, as shown in Figure 8. This is due to the fact that I/O bandwidth is shared thus not stable. When we do the profiling, we get the writing speed in Bebop is around 2 GB/s. However, when the data is dumped, the actual writing speed is only 1.05 GB/s, which leads to the wrong decision for discarding the lossless compression stage

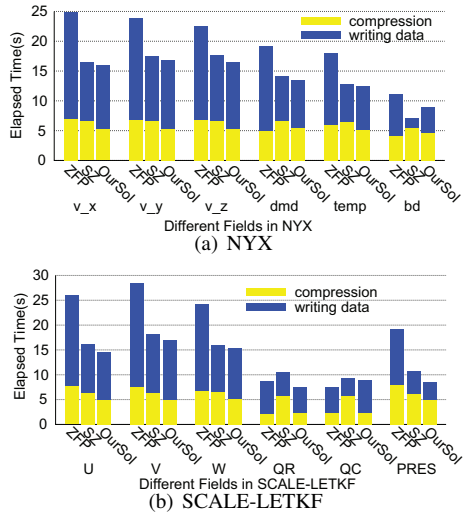


Fig. 8. Detailed breakdown of dumping performance on each field (error bound 1E-6) on Bebop with 4 nodes (128 cores)

in our solution. However, this situation only occurs around the turning point, which would not incur much overhead even when the decision is wrong (even in this case when the writing speed is inaccurate by 50%). Generally speaking, our solution would work better in a relatively stable environment like Theta, as will be discussed in next section.

C. Dumping performance on Theta

We perform similar experiments on Theta. Unlike Bebop, the turning point for choosing SZ and ZFP is around 256 $\sim$ 512 nodes. Therefore, we show the result on Theta with 256 nodes and 512 nodes, respectively. Please note that we pad the compressed data to multiple of 64KB and set the maximal padded size as the stripe size (which is the real compressed size) for best performance, as required by the Lustre file system on Theta.

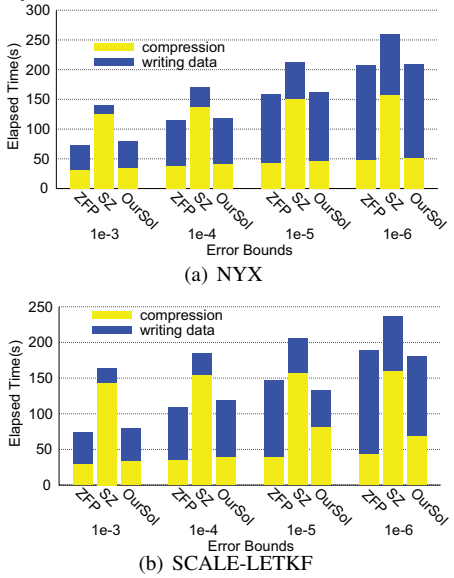


Fig. 9. Dumping performance of different error bounds (1E-3 $\sim$ 1E-6) on Theta with 256 nodes (16,384 cores)

The overall dumping performance on Theta with 256 nodes (16,384 cores) is displayed in Figure 9. In this case, the total

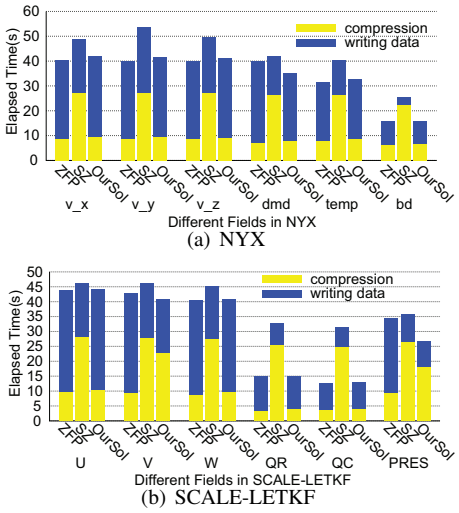


Fig. 10. Detailed breakdown of dumping performance on each field (error bound 1E-6) on Theta with 256 nodes (16,384 cores)

data to be dumped in a single snapshot would be 48TB and 51TB, costing ∼410s and ∼440s on data writing, respectively. For each dataset, the writing bandwidth is 115GB/s according to Figure 1). In this situation, the compression time for SZ would easily become larger than the writing time, as shown in this figure. Then, our adaptive framework would prefer ZFP in most cases, as shown in the detailed performance breakdown on each field in Figure 10. From this figure, we can see that our solution always selects ZFP for the NYX dataset, leading to similar execution time compared with ZFP. Please note that our solution would have a low overhead due to the sampling and synchronization process. It is around 5% when the error bound is 1E-3 and reduces to less than 2% when the error bound is 1E-6 on the two datasets, due to the fact that the percentage of compression time over total dumping time would decrease with the decreasing error bound (low error bound leads to low compression ratio thus high writing time). Nevertheless, our solution outperforms ZFP when the error bound is tight, since our framework accurately picks four fields (the first, second, third and last) out of the six fields in which the optimized SZ is faster than ZFP considering the overall dumping performance.

We then show the result with slightly increased scale in Figure 11, by using 512 nodes (32,768 cores) on Theta. Note that the experiments here are all weak-scaling thus the file size to be dumped is doubled. Compression ratios become more important in this case because of the increased time on writing, especially when the error bound is relatively small. In this case, our adaptive solution would select the optimized SZ for most of the time. As shown in Figure 12, it chooses to use the optimized SZ for five out of six fields in NYX and four out of six fields in SCALE-LETKF, when the error bound is set to 1e-5. This decision leads to over 20% performance gain in these fields (e.g. the first, second, third and fifth field in NYX and the first, second, third and the last field in SCALE-LETKF). As these fields dominate the total dumping time, the final performance gains for our solution are 20% and 27%, respectively, compared to the second-best compressor for the two datasets in this case.

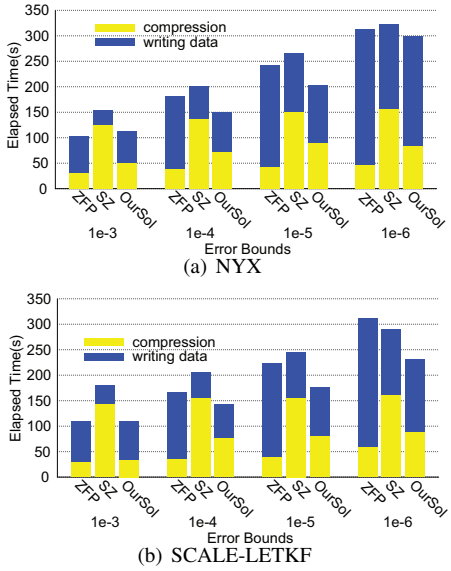


Fig. 11. Dumping performance of different error bounds (1E-3 ∼ 1E-6) on Theta with 512 nodes (32,768 cores)

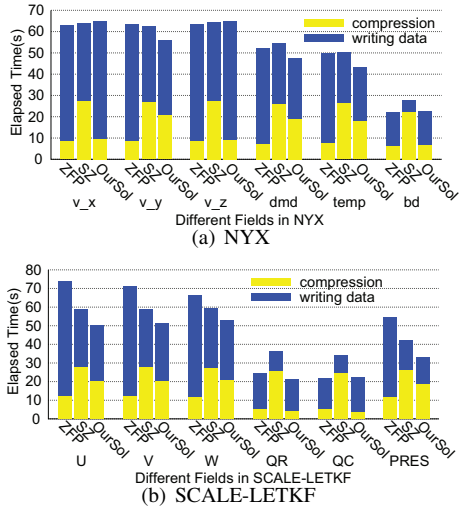


Fig. 12. Detailed breakdown of dumping performance on each field (error bound 1E-6) on Theta with 512 nodes (32,768 cores)

VII. Conclusion and Future Work

In this paper, we design an adaptive lossy compression framework selecting best-fit lossy compressor and optimal compression settings for different scales at runtime. The proposed solution achieves up to 27% performance gain over other approaches on Theta with 32k cores. In the future, we plan to improve data loading performance by optimizing the tradeoff between decompression time and compression ratio.

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, under contract DE-AC02-06CH11357, and supported by the National Science Foundation under Grant No. 1619253. This work was also supported by National Science Foundation CCF 1513201. We acknowledge the computing resources provided on Bebop, which is operated by the Laboratory Computing Resource Center at Argonne National Laboratory.

REFERENCES

[1] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, et al., "Simulating sky surveys on state-of-the-art supercomputing architectures," in *New Astronomy*, 42:4965, 2016.

[2] V. Vishwanath, S. Crusan and K. Harms, "Parallel I/O Performance", [Online]. Available at: https://www.alcf.anl.gov/files/Parallel_IO_on_Mira_0.pdf

[3] N. Sasaki, K. Sato, T. Endo, and S. Matsuoka. 2015. "Exploration of lossy compression for application-level checkpoint/restart," In 2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS). 914922.

[4] P. Lindstrom. Fixed-Rate Compressed Floating-Point Arrays. *IEEE Trans. on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.

[5] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Ku, C. Chang, S. Klasky, R. Latham, R. B. Ross, and N. F. Samatova, "ISABELA for effective in situ compression of scientific data," in *Concurrency and Computation: Practice and Experience*, 25(4):524540, 2013.

[6] P. Lindstrom and M. Isenburg, "Fast and Efficient Compression of Floating-Point Data," in *IEEE Trans. on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1245–1250, 2006.

[7] S. Di and F. Cappello, "Fast Error-bounded Lossy HPC Data Compression with SZ," In *Proceedings of IEEE 30th International Parallel and Distributed Processing Symposium (IPDPS16)*, pp. 730–739, 2016.

[8] 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 *IEEE International Parallel and Distributed Processing Symposium IPDPS2017*, Orlando, Florida, USA, May 29-June 2, pp. 1129–1139, 2017.

[9] 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 IEEE International Conference on Big Data, 2018.

[10] S. Li, S. Di, X. Liang, Z. Chen and F. Cappello, "Optimizing Lossy Compression with Adjacent Snapshots for N-body Simulation Data", In IEEE International Conference on Big Data, 2018.

[11] X. Liang, S. Di, D. Tao, Z. Chen and F. Cappello, "An Efficient Transformation Scheme for Lossy Data Compression with Point-wise Relative Error Bound", In IEEE International Conference on Cluster Computing, 2018.

[12] Theta supercomputer. [Oline]. Available at https://www.alcf.anl.gov/theta

[13] Gzip compression. [Online]. Available at http://www.gzip.org.

[14] BlosC compressor. [Online]. Available at http://blosc.org

[15] M. Burtscher and P. Ratanaworabhan, "High Throughput Compression of Double-Precision Floating-Point Data," In *Data Compression Conference (DCC'07)*, pp. 293–302, 2007.

[16] F. Cappello, S. Di, S. Li, X. Liang, A. M. Gok, D. Tao, C. H. Yoon, X. Wu, Y. Alexeev, and F. T Chong, "Use Cases of Lossy Compression for Floating-Point Data in Scientific Data Sets," in *The International Journal of High Performance Computing Applications*, 2019.

[17] S. Ziegeler, C. Stone. "Synthetic Data Generation for Evaluating Parallel I/O Compression Performance and Scalability," in The 3rd International Workshop on Data Reduction for Big Scientific Data, 2018.

[18] S. Madireddy et al., "Modeling I/O Performance Variability Using Conditional Variational Autoencoders", CLUSTER'18: The 2018 IEEE International Conference on Cluster Computing, pp. 109-113, Belfast (2018)

[19] Madireddy S. et al., "Machine Learning Based Parallel I/O Predictive Modeling: A Case Study on Lustre File Systems". In: ISC'18: International Conference on High Performance Computing (2018)

[20] Tran, N., Reed, D.A.: "ARIMA time series modeling and forecasting for adaptive i/o prefetching". In: ICS'01: The 15th International Conference on Supercomputing. pp. 473–485. Sorrento, Italy (2001)

[21] Tran, N., Reed, D.A.: "Automatic arima time series modeling for adaptive i/o prefetching. IEEE Transactions on Parallel and Distributed Systems" 15(4), pp. 362–378 (2014)

[22] Q. Noorshams, K. Rostami, S. Kounev and R. Reussner. Modeling of I/O Performance Interference in Virtualized Environments with Queueing Petri Nets. In: MASCOTS'14: IEEE 22nd International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, Paris, pp. 331-336 (2014)

[23] Y. Wu, F. Ye, K. Chen and W. Zheng. Modeling of Distributed File Systems for Practical Performance Analysis. In: IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 1, pp. 156-166 (2014)

[24] Glenn K. Lockwood, Shane Snyder, Teng Wang, Suren Byna, Philip Carns, and Nicholas J. Wright. A year in the life of a parallel file system. In SC'18: International Conference for High Performance Computing, Networking, Storage, and Analysis. pp. 74:1–74:13 (2018)

[25] T. Wang, S. Snyder, G. Lockwood, P. Carns, N. Wright and S. Byna. IOMiner: Large-Scale Analytics Framework for Gaining Knowledge from I/O Logs. In CLUSTER'18: The 2018 IEEE International Conference on Cluster Computing, Belfast, pp. 466-476 (2018)

[26] Top500 list. [Online]. Available at https://www.top500.org/lists/2018/11/.

[27] T. Lu, Q. Liu, et al. "Understanding and Modeling Lossy Compression Schemes on HPC Scientific Data," in *32nd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2018)*, 2018.

[28] Zstd. Available at: https://github.com/facebook/zstd/releases

[29] D. Tao, S. Di, X. Liang, Z. Chen and F. Cappello, "Optimizing lossy compression date-distortion from automatic online selection between SZ and ZFP", to appear in IEEE Transactions on Parallel and Distributed Systems (TPDS), online at https://arxiv.org/abs/1806.08901, 2018

[30] L. Ibarria, P. Linderstrom, J. Rossignac, and A. Szymczak, "Outofcore compression and decompression of large ndimensional scalar fields," in *Computer Graphs Forums*, vol. 22, no. 3, pages 343–348, 2003.

[31] Bebop supercomputer. [Online]. Available at https://www.lcrc.anl.gov/systems/resources/bebop.

[32] Lustre file system. [Online]. Available at: http://lustre.org/.

[33] R. Thakur, W. Gropp, E. Lusk, "On implementing MPI-IO portably and with high performance," Proceedings of the sixth workshop on I/O in parallel and distributed systems, pages 23–32, 1999.

[34] NYX simulation. [Online]. Available at: https://amrex-astro.github.io/Nyx/.

[35] SCALE-LETKF weather model. [Online]. Available at: https://github.com/gylien/scale- letkf.