Compression Ratio Modeling and Estimation across Error Bounds for Lossy Compression

Jinzhen Wang, Tong Liu, Qing Liu, Xubin He, *Senior Member, IEEE*, Huizhang Luo, and Weiming He

Abstract—Scientific simulations on high-performance computing (HPC) systems generate vast amounts of floating-point data that need to be reduced in order to lower the storage and I/O cost. Lossy compressors trade data accuracy for reduction performance and have been demonstrated to be effective in reducing data volume. However, a key hurdle to wide adoption of lossy compressors is that the trade-off between data accuracy and compression performance, particularly the compression ratio, is not well understood. Consequently, domain scientists often need to exhaust many possible error bounds before they can figure out an appropriate setup. The current practice of using lossy compressors to reduce data volume is, therefore, through trial and error, which is not efficient for large datasets which take a tremendous amount of computational resources to compress. This paper aims to analyze and estimate the compression performance of lossy compressors on HPC datasets. In particular, we predict the compression ratios of two modern lossy compressors that achieve superior performance, SZ and ZFP, on HPC scientific datasets at various error bounds, based upon the compressors' intrinsic metrics collected under a given base error bound. We evaluate the estimation scheme using twenty real HPC datasets and the results confirm the effectiveness of our approach.

Index Terms—High-performance computing, lossy compression, data reduction, performance modeling

1 Introduction

Data reduction has become increasingly important for simulation-based scientific discovery. For large datasets generated from high-performance computing (HPC) simulations, data reduction techniques aim to lower data volume and velocity so that the overhead of I/O and data analysis is more tractable. In general, data reduction takes advantage of the inherent redundancy in data, and state-of-the-art floating-point compressors can be either lossless [1], [2], [3], [4] or lossy [5], [6], [7], [8], [9], [10], [11], depending on whether there is information loss during compression. While lossless compression incurs no information loss, the resulting reduction performance is often mild and far from being sufficient when dealing with extreme-scale datasets, e.g., those of petabytes produced by simulations running at scale. Lossy compression, by trading accuracy for performance, offers a much higher reduction performance, e.g., a 400X of compression ratio as reported in prior work [7], to mitigate the bottleneck of I/O.

In reality, information loss is commonly accepted and leveraged by domain scientists to lower the application complexity. A classic example is the particle-in-cell (PIC) numerical solver, in which the amount of macro-particles injected to the system is orders of magnitude less than that of physical particles, thus greatly reducing the computational complexity [12]. Therefore, for large-scale data management, lossy compression is often the preferred path forward due to its high reduction performance. When using lossy compressors, such as SZ [7] and ZFP [6], domain scientists are required to specify an error bound (or precision) in order to control the loss of accuracy of their data. A key challenge is that without fully understanding the complex interplay between error bound of their choice and the reduction performance, choosing an appropriate error bound is often difficult and can only be done through trial and error.

Intuitively speaking, there exists a positive correlation between error bound and compression ratio. That is, the looser the error bound is, the higher compression ratio can be achieved. Prior work [13] has shown that a looser error bound in SZ can result in a higher hit ratio of curve-fitting, which in turn improves the compression ratio. However, this may not be true across all error bounds. Fig. 1 shows such relationships in SZ and ZFP, respectively, on twenty scientific datasets. For ZFP, while it is clear that the compression ratio increases monotonically as the error bound loosens, the trend cannot be characterized by a simple linear or exponential relationship. Meanwhile, for SZ, the monotonically increasing trend is only applicable to some datasets, e.g., *Bump*, *Sedov*, *NWChem*, *QMPACK*, *HACC*, *XGC* and *S3D*. For others, the compression ratio decreases slightly initially, and then quickly ramps up. With such counter-intuitive trends of compression ratios, it is hard for domain scientists to select an appropriate error bound that satisfies their reduction goals while minimizing the loss of information. They may have to exhaust many error bounds before identifying a satisfactory one, and this

- *J. Wang, Q. Liu, H. Luo, and W. He are with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102. E-mail: {jw447, qing.liu, huizhang.luo, wh249}@njit.edu.*
- *T. Liu and X. He are with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122. E-mail: {tongliu, xubin.he}@temple.edu.*

Manuscript received 29 Dec. 2018; revised 15 Aug. 2019; accepted 17 Aug. 2019. Date of publication 30 Aug. 2019; date of current version 3 Mar. 2020. (Corresponding author: Qing Liu.)
Recommended for acceptance by B. He.
Digital Object Identifier no. 10.1109/TPDS.2019.2938503

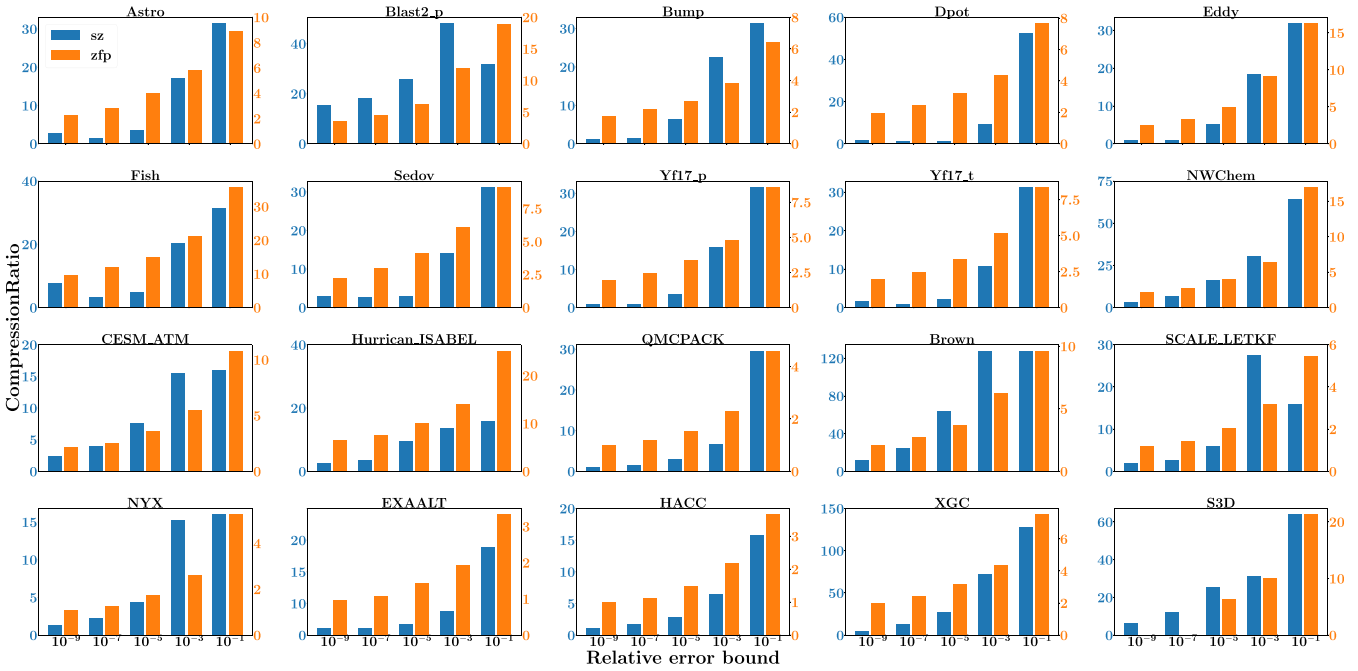


Fig. 1. Compression ratio versus error bound (SZ and ZFP).

process can be both time- and resource-consuming for large datasets, since compression is highly computation-intensive in nature.

To that end, this paper aims to gain a deep understanding of the inner mechanisms of lossy compressors, as well as develop modeling techniques to guide the selection of appropriate error bounds. Our goal is to predict the order of magnitude of compression ratios so that the estimation can be useful in science production in order to satisfy the storage constraints. Our approach is built upon a key observation that the internal metrics of a lossy compressor are highly correlated with error bound and thus predictable across error bounds (Section 3). Therefore, by collecting these compression metrics at a particular error bound, termed as *base error bound* (denoted as EB_{base}), one can quantitatively understand the compression behavior for a given dataset, and extrapolate the compression ratio to another error bound (denoted as EB_{new}). This idea is implemented and verified against two leading floating-point lossy compressors, ZFP and SZ, which have shown to be superior among all state-of-the-art solutions. The results demonstrate that our models can estimate compression ratios at EB_{new} with decent accuracies. Ultimately, we envision that for large-scale simulations, we can downsample their data while in memory, predict the compression ratios at EB_{base}, as demonstrated in prior work [13], and further extrapolate compression ratios from EB_{base} to all other error bounds. This allows domain scientists to estimate compression ratios with greatly reduced overhead relative to trial and error and thereby choose the error bound that best fits their needs prior to compression.

The contributions of our work are summarized as follows:

- We experimentally demonstrate that the internal compression metrics are correlated with error bounds and thus predictable across error bounds.
- We propose a modeling methodology to predict the compression ratios at EB_{new} based upon compression metrics collected at EB_{base}. We further implement our techniques for ZFP and SZ, respectively, which adopt very different design strategies for compression, thus, resulting in different compression models.
- We evaluate our modeling techniques using twenty real scientific datasets and demonstrate the accuracy of our techniques.

We note that for SZ, our work focuses on 1D data compression using the latest stable version (SZ 2.0) in experiments, since all our datasets are linearized to 1D first across the board for consistency. As a result, the linear regression prediction primarily developed for multi-dimensional compression is not activated during compression. More discussions on the this can be found in the section of related work. The rest of the paper is organized as follows. Section 2 discusses related work, Section 3 introduces the motivation, and Section 4 investigates the modeling and prediction of compression ratios between two error bounds. Specifically, we first discuss the estimation methodology and then implement the cross error bound modeling on SZ and ZFP. In Section 5, we evaluate our models by comparing the prediction results with the real compression ratios, along with conclusions in Section 6.

2 Related Work

Modern floating-point compressors fall into two categories, lossless compressors [1], [2], [3], [4] and lossy compressors [5], [6], [7], [8], [9], [10], [11]. Lossless compressors, such as FPC [1], FPZIP [3], and GZIP [4], are typically used in scenarios where information loss is not tolerable, e.g., compressing checkpoints. They often provide limited data reduction due to the high entropy of floating-point data.

Compared to lossless compressors, error-bounded lossy compressors can provide orders of magnitude higher compression ratios. Motivated by the reduction potential of spline functions [14], [15], ISABELA [5] is designed to compress spatial-temporal scientific data with huge randomness and noise; it sorts multi-dimensional floating-point data to make them smoother and more compressible. However, the sorting operation loses locality information, and therefore ISABELA needs to maintain an extra index which hurts the performance of reduction. Deering [9] introduces a mesh compression algorithm to compress 3D triangle data. It utilizes triangular meshes, delta compression, and a modified Huffman compression to compress vertex locations, colors, and normals of 3D triangle data. It reduces the storage of numeric information about each point in a mesh. Cohen-Or et al. [10] presents a progressive mesh compression method. Based on multi-resolution decomposition, it offers different resolutions given current network conditions with regard to the bandwidth and transmission latency. This is particularly useful if the network condition is poor. ZFP [6], is a block transformation based compression algorithm. It takes a block of 4^d points at a time, where d is the number of dimensions. It offers supreme compression performance for multi-dimensional data and enables random access to any block of data, since each block is individually compressed. Austin et al. [8] uses a Tucker decomposition based method to compress extreme-scale data in parallel. The results show that the parallel decomposition can achieve high compression ratios and be easily scaled to high core counts.

The initial version of SZ 0.1 [7] was released in 2016. It flattens multi-dimensional data into a single-dimensional array and exploits the local smoothness within an array for compression. It uses multiple curve-fitting schemes to predict a data point from preceding points. The curve-fitted points are further compressed using Huffman encoding, while the curve-missed points are compressed by the binary representation analysis. SZ 1.4 [16] further improves the compression ratio of multi-dimensional data through enhancing the curve-fitting prediction. In this version, multi-dimensional data is no longer flattened into single dimension first but curve-fitted using a multi-dimensional prediction method via a linear-scaling quantization. Furthermore, SZ 2.0 [17] improves the compression quality at high compression ratio cases by introducing an adaptive selection method between the mean-integrated Lorenzo predictor and a linear regression-based predictor. The linear regression-based predictor is shown to be more accurate for data that follow the normal distribution while the mean-integrated Lorenzo predictor is more effective if data is non-smooth.

Due to the distinct characteristics of applications as well as data in the HPC domain, the general-purpose compression often cannot satisfy all needs. As a result, improvements and modifications have also been made on a per application basis. For efficient and low-overhead compression of quantum chemistry data, SZ was recently improved to leverage latent data features and optimized on the number of bits for storing the two-electron repulsion integrals [18]. Further, in order to mitigate the bottlenecks on memory and network for high-resolution multi-dimensional visualization, a tensor decomposition based lossy compression [19] is introduced to use the higher-order singular value decomposition (HOSVD) to achieve high reduction ratios. In our prior work Canopus [11], a progressive data management scheme is designed to store and analyze extreme-scale scientific data. It co-designs data decimation, compression and storage, mapping data to different storage tiers with different latency levels. A key advantage of Canopus is that users can decide the trade-off between analysis speed and data accuracy, and allows analysis to work on lower-accuracy data and augment its accuracy only if needed.

Despite the recent success in this area, there is still a lack of understanding of the interplay between reduction performance and various design parameters. In our prior work [13], the relationship between reduction performance and error tolerance has been briefly discussed. It was observed that the compression ratio of a lossy compressor increases as the error bound loosens. This trend generally holds for ZFP, but it displays some inconsistency with SZ when the error bound is tight. The work developed an analytical model to predict the compression ratio of a full dataset from a sampled dataset. The model reduces the overhead of processing large datasets while maintaining low estimation error. It is evaluated that when sampling at the ratio of 1 and 10 percent, the estimation error is at 15.7 and 6.9 percent, respectively, for the *astro* dataset. However, in reality, when dealing with HPC datasets at the scale of petabytes, 1 percent of sampling will still result in a data volume that is expensive to process.

A recent work by Tao et al. [20] developed a lightweight online selection tool to choose between SZ and ZFP during compression. It estimates the compression ratios of SZ and ZFP under the same peak signal-to-noise ratio (PSNR) on a uniformly sampled subset of the original dataset. Specifically, it computes the probability density function (PDF) of quantization factor distribution on sampled dataset to approximate the bit-rate of SZ. For ZFP, the size of each block is estimated using the number of significant bits on sampled data. Then the compressor that yields a higher compression ratio is selected for compression. In comparison, our work has a different goal of estimating the compression ratio across error bound, based upon the compression metrics collected at the base error bound only.

3 Motivation

Our previous work [13] first proposed a sampling-based methodology to predict compression ratios of SZ and ZFP. The key idea is to use sampled data to extrapolate the compression ratios of the full data, leveraging the statistical similarity between the two datasets. Despite the fact that the estimation scheme achieves a high accuracy, the outcome of the estimation is sensitive to the sampling ratio. With a higher sampling ratio (thus smaller sampled data), more information is lost, and thus the estimation will deviate from the true compression ratio. Conversely, with a lower sampling ratio, such as 1 and 0.1 percent, the scheme can achieve fair estimations. However, extreme-scale datasets at the level of petabytes will still be very costly to handle after being downsampled to terabytes. Further, to estimate the compression ratios at multiple error bounds, one has to compress the downsampled data at each of the target error

TABLE 1
SZ Compression Metrics Across Error Bounds

Compression Metric	EB_1	EB_2	EB_3	EB_4	Correlation w. log of error bound
Errorbound	10^{-9}	10^{-7}	10^{-5}	10^{-3}	1.00
NodeCount	20,551	40,421	13,383	1,467	−0.66
HitRatio	0.78	0.98	1.0	1.0	0.82
Quantization Factor	20,551	40,421	13,383	1,467	−0.66
Mean of quantization factor	798,747	1,024, 573	1,048, 574	1,048, 576	−0.87
Variance of quantization Factor	207,742, 027,628	30,042, 598,502	37,446, 718	3,770	0.87
TreeSize	184,960	363,790	120,448	13,204	−0.66
EncodeSize	49,900	70,639	50,503	27,354	−0.64
OutlierSize	74,205	4,623	0	0	−0.80

EB_1 to EB_4 are Four Error Bound Configurations. The detailed description of these metrics can be found in Table 3.

bounds, which would be too expensive. At last, estimating the compression ratio of the full dataset from the sampled dataset is not always feasible, since this approach strictly requires the bounded locality [21].

In light of the issues above, we take a new direction to achieve compression ratio estimation across error bounds. Our approach is motivated by the correlation between compression metrics and error bounds. We show the compression metrics of SZ and ZFP on the *Dpot* dataset followed by the Pearson correlation with the logarithm of error bound in Tables 1 and 2, respectively. The detailed explanation of these compression metrics can be found in Sections 4.3.1 and 4.4.1.

For SZ, most compression metrics, e.g., *HitRatio*, *OutlierSize*, *Mean of quantization factor* and *Variance of quantization factor* are highly correlated to the error bound. However, *NodeCount*, *TreeSize*, *EncodeSize*, *OutlierSize* and *QuantizationFactor* are less correlated to the error bound. This is because *NodeCount* is observed to increase first and decrease later with the error bound, thus being less correlated with the error bound, and it is a dominating factor that in turn affects *TreeSize*, *EncodeSize*, *OutlierSize*, and *QuantizationFactor*.

Similarly, for ZFP, the compression metrics with Pearson correlation coefficients to the logarithms of error bounds are shown in Table 2. It is shown that all parameters are highly correlated to the error bound, except *MaxExp* which is constant across the error bound. We note that the Pearson correlation coefficient is undefined for a random variable that zero variance.

TABLE 2
ZFP Compression Metrics Across Error Bounds

Compression Metric†	EB_1	EB_2	EB_3	EB_4	Correlation w. log of error bound
Errorbound	10^{-9}	10^{-7}	10^{-5}	10^{-3}	1.00
BitsPerBitplane	3.50	3.48	3.44	3.37	−0.98
MaxPrec	33.84	26.84	19.84	13.84	−1.0
MaxExp	2.84	2.84	2.84	2.84	N/A
BlockSize	118.53	93.34	68.15	46.56	−0.99

† *Since ZFP compresses each block of data individually, instead of showing the compression metrics for all blocks, we show the average value. The correlation coefficient is undefined for* MaxExp, *since it has zero variance.*

TABLE 3
A List of Symbols

Symbols	Description
	General
InputSize	Size of uncompressed dataset
OutputSize	Size of compressed dataset
CompressionRatio	The ratio of *InputSize* to *OutputSize*
EB_{base}, EB_{new}	base error bound and target error bound to predict *CompressionRatio*
	SZ compression metrics
NodeCount	Number of Huffman tree nodes
HitRatio	Curve-fitting hit ratio
QuantizationFactors	Number of quantization factors used in Huffman encoding
TreeSize	Size of Huffman tree structure (in bytes)
EncodeSize	Size of Huffman encoding for all nodes (in bytes)
OutlierSize	Size of binary representation of curve-missed points (in bytes)
qf_ebase_min, *qf_ebase_max*	The smallest and largest quantization factor at EB_{base}
qf_enew_min, *qf_enew_max*	The smallest and largest quantization factor at EB_{new}
	ZFP compression metrics
BitsPerBitplane	Number of bits used in encoding each bit plane
MaxPrec	Maximum number of bit planes to encode in order to meet the accuracy demand
MaxExp	The common (largest) exponential of each block
BlockSize	Size of each block data (in bits)

This observation motivates us to leverage the correlation and capture the trend of compression metrics in order to estimate the compression ratio. We next discuss the methodology of capturing the trend of compression metrics and compression ratio estimation.

4 Compression Ratio Estimation Across Error Bounds

In this section, we first discuss the general methodology of estimating compression ratios across error bounds. We then develop the prediction models for SZ and ZFP. For both compressors, our approach is to first predict the internal compression metrics across error bounds and then estimate compression ratios.

4.1 Basics

For the convenience of discussion, we list the notations used in the paper in Table 3. The datasets used for evaluation are briefly described in Table 4. The approach here is that by compressing data once at EB_{base} and additionally collecting a small set of compression metrics, one can capture the characteristics of data and behavior of a compressor, as well as the interplay between them. Based on this, we can further assess the compression ratio at EB_{new}. The compression

TABLE 4
Dataset Description

Dataset	Dimension and type	Size	Description
Dpot	1×20694, double precision	166 KB	Electric potential deviation in a plasma physics simulation
Astro	1×65536, double precision	524 KB	Velocity magnitude in a supernova simulation
Fish	1×65536, double precision	524 KB	Velocity magnitude in a CFD calculation of cooling air being injected into a mixing tank
Sedov	1×78144, double precision	625 KB	Pressure of strong shocks in a hydrodynamical simulation
Blast2_p	1×578880, double precision	5 MB	Pressure of strong shocks in a gas-dynamical simulation
Eddy	1×282616, double precision	2 MB	Velocity in a 2D solution to Navier-Stokes equations
Yf17_p	1×97104, double precision	777 KB	Pressure in a computational fluid dynamics calculation
Yf17_t	1×97104, double precision	777 KB	Temperature in a computational fluid dynamics calculation
Bump	1×55692, double precision	446 KB	Flow density of an axisymmetric bump
CEMS_ATM	$26 \times 1800 \times 3600$, single precision	674 MB	Climate simulation
EXAALT	1×2869440, single precision	12 MB	Molecular dynamics simulation
Hurricane_ISABEL	$100 \times 500 \times 500$, single precision	100 MB	Climate simulation of hurricane
HACC	1×280953867, single precision	1 GB	Cosmology: particle simulation
NYX	$1 \times 512 \times 512$, single precision	537 MB	Cosmology: Adaptive mesh hydrodynamics + N-body cosmological simulation
NWChem	1×102953248, double precision	824 MB	Two-electron repulsion integrals computed over Gaussian-type orbital basis sets
QMCPACK	$115 \times 69 \times 69 \times 288$, single precision	631 MB	Many-body ab initio Quantum Monte Carlo (electronic structure of atoms, molecules, and solids)
S3D	$500 \times 500 \times 500$, double precision	11 GB	Combustion simulation
XGC	20694×512, double precision	339 MB	Fusion Simulation
Brown	1×8388609, double precision	268 MB	Synthetic, generated to specified regularity
SCALE_LETKF	$98 \times 1200 \times 1200$, single precision	565 MB	Climate simulation

ratio, denoted as *CompressionRatio*, is defined as the ratio of original data size, *InputSize*, to the compressed data size, *OutputSize*, as shown in Equation (1). Clearly, for a given dataset, the problem of modeling *CompressionRatio* comes down to the modeling of *OutputSize*.

$$CompressionRatio = \frac{InputSize}{OutputSize}. \quad (1)$$

As aforementioned, the error bound EB_i controls the tolerance of information loss during data compression. In general, there are two types of error bounds, *absolute* and *relative* error bounds, that are widely used in HPC data compression. Assume a data point has a value denoted as V, the absolute error bound is an upper bound of the difference between the original value and the decompressed value, so the decompressed value is in the range of $[V - EB_i,\ V + EB_i]$. In contrast, the relative error bound allows for an error that is relative of V and has an error tolerance range of $[V \cdot (1 - EB_i),\ V \cdot (1 + EB_i)]$. Unless otherwise specified, we adopt the relative error bound in our work, since it results in commensurate information loss for both high and low values.

4.2 Methodology

To predict the compression ratio at error bound EB_{new}, we first perform a standard compression at error bound EB_{base}. During this process, we collect a set of compression metrics (detailed in Table 3) that have shown to be correlated with error bounds (Section 3). For example, we experimentally observed in SZ that the distribution of quantization factors, a key intermediate compression product that is more compressible than the original data, are highly similar at different error bounds and can be approximated by the Gaussian distribution. This critical observation enables us to extrapolate the quantization factors from EB_{base} to EB_{new} (Section 4.3.1). Once the quantization factors are obtained at EB_{new}, we can further construct the new Huffman tree and extrapolate the compression ratio. Also, for ZFP, due to its block-wise operation (Section 4.4.1), characterizing the number of bits used per each block via two parameters, the number of bit planes to encode and the average bits used to encode each bit plane, will allow us to predict the average block size at other error bounds. The general methodology is described as follows:

- *Step 1:* We run a standard compression at EB_{base} and collect a set of internal compression metrics, which are compressor dependent.
- *Step 2:* We analyze the compression metrics at EB_{base} and build models to predict them at EB_{new}. The intuition behind this is that by capturing the compression metrics, we can indirectly understand the data characteristics as well as how a compressor reacts to the data. These can be further exploited to extrapolate the compression performance.
- *Step 3:* We use the estimated compression metrics to further predict the compression ratio at EB_{new}.

Next, we discuss the estimation of SZ and ZFP, respectively.

4.3 Prediction of SZ Compression Ratio

4.3.1 SZ Compression and Its Internal Metrics

In this work, we focus on a recent version of SZ 2.0. Since some of the datasets have already been linearized, we compress all datasets as one-dimensional across the board for consistency. Therefore, the proposed regression-based

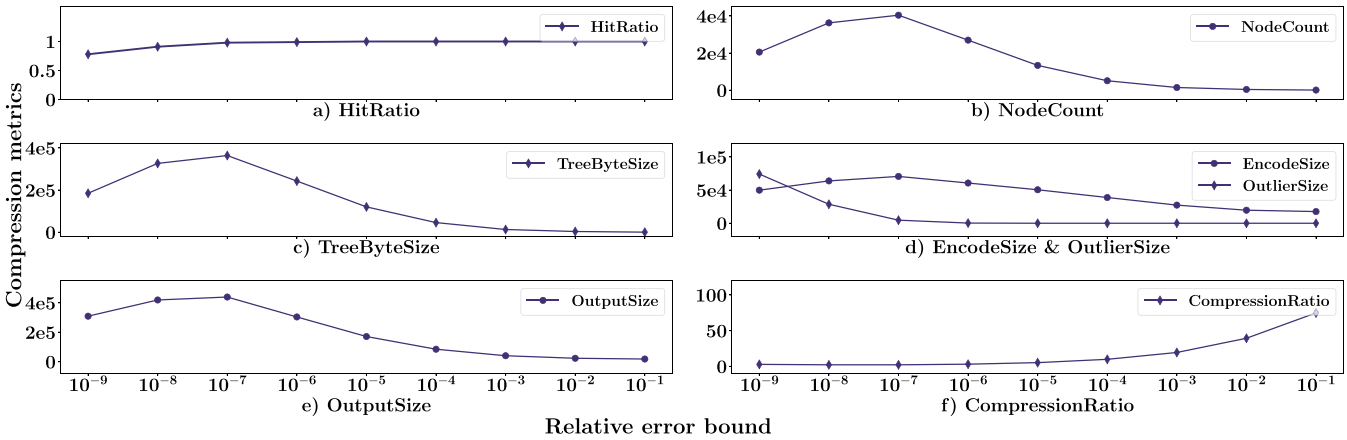


Fig. 2. SZ compression metrics versus relative error bound (*Astro*).

prediction model is not used in our work since it targets multi-dimensional compression. For each data point, SZ checks whether it can be predicted by its previous points using either linear or quadratic curve-fitting, subject to a user specified error bound. If so, this data point is deemed to be curve-fitted and is further discretized using a quantization factor followed by Huffman encoding. The intuition is that if there is local smoothness in data, the likelihood that data points are distributed closely around the predicted value is high. Therefore, after quantization, data points could potentially be mapped to an identical quantization factor and thus can be further compressed using Huffman encoding. The number of quantization factors, denoted as *QuantizationFactor*, is the number of discrete levels that SZ maps a curve-fitted data value into. This parameter can be either prescribed by the user or calculated by the compressor based on the data range and error bound. If the data point cannot be predicted by its previous points, it is deemed to be curve-missed and is encoded using binary representation analysis. Namely, it utilizes *IEEE format 754* to represent curve-missed data points where data values are normalized and truncated based on the error bound, and then optimized by a leading-zero based compression method [7].

As such, the size of SZ-compressed data consists of Huffman tree size, Huffman encoding size for curve-fitted data points, and binary representation size for curve-missed data points, as shown in Equation (2). We next first analyze the impact of each individual component on *OutputSize*.

$$OutputSize = TreeSize + EncodeSize + OutlierSize. \quad (2)$$

We observe that most of the compression power in SZ comes from the curve-fitting and Huffman encoding. For example for *Astro*, as shown in Fig. 2, when the relative error bound is higher than 10^{-8}, *HitRatio* is consistently above 90 percent. Thus, the majority of data points are hit by curve-fitting. Therefore, we focus on curve-hitting and Huffman encoding in what follows.

HitRatio. A key compression metric that measures the effectiveness of curve-fitting is *HitRatio*, which is the percentage of data points that can be curve-fitted and encoded using the Huffman tree under a given error bound. In Fig. 2a, we observe that for *Astro*, *HitRatio* increases monotonically from 70 percent to approximately 100 percent when the error bound loosens from 10^{-9} to 10^{-1}. Intuitively, the associated Huffman tree quantities *TreeSize* and *EncodeSize* should also increase since more data points need to be encoded as a result of increasing *HitRatio*. Nevertheless, the results in Figs. 2c and 2d show that *TreeSize* and *EncodeSize* increase at first and then drop after the error bound reaches 10^{-7}. Thus, despite being an important metric, *HitRatio* may not be the sole factor in determining the outcome of compression.

NodeCount. Here *NodeCount* is the number of Huffman tree nodes used to encode the quantization factors. We observe that the resulting *TreeSize* and *EncodeSize* follow a similar trend as *NodeCount* across error bounds, indicating that *NodeCount* is another key factor that affects compression. Namely, it increases at first and then drops when error bound reaches 10^{-7}, which is also the point where *HitRatio* reaches 100 percent (Fig. 2b). Since *NodeCount* is equivalent to the unique number of quantization factors used in Huffman encoding, we aim to study the distribution of quantization factors across different error bounds and understand the trend of *NodeCount*. It can be seen from Fig. 3 that quantization factors exhibit similar shapes across error bounds from 10^{-9} to 10^{-7}, but the shape narrows drastically thereafter. Namely, the range of quantization factors decreases from $[0, 2497152]$ to $[1048540, 1048612]$, and the number of points represented by each factor, indicated by the bar height, increases. We observe that fewer quantization factors are used after *HitRatio* approximates to 100 percent. The reason is that no more data points can be curve-fitted, and further loosening error bound will result in more data points being covered by a single quantization factor. The decreasing of unique quantization factors further leads to the decreasing of *NodeCount*.

Therefore, we believe *HitRatio* and *NodeCount* are the two main metrics affecting the compression of SZ. To extrapolate the compression ratio, we need to first model *HitRatio* and *NodeCount*, respectively.

4.3.2 SZ Modeling and Estimation

In this section, we aim to predict SZ compression metrics from the base error bound, EB_{base}, to another error bound, EB_{new}, and further predict $CompressionRatio_{new}$—the compression ratio at EB_{new}. To this end, we first discuss the modeling of *HitRatio* and *NodeCount*.

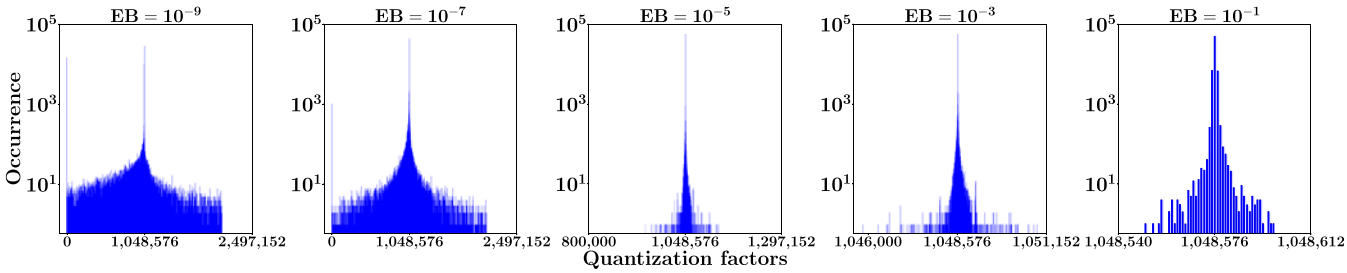


Fig. 3. Quantization factor distribution (*Astro*). Each plot shows the histogram of quantization factors under a particular error bound. The x-axis is the value of quantization factor, and the y-axis is the count of occurrence. The title of each plot is the relative error bound used.

HitRatio. For SZ, *HitRatio* can be fairly well predicted, since whether a data point is a hit or a miss only involves a simple comparison between the prediction error and the radius of hit, denoted as *HitRadius*, which is calculated as $HitRadius = EB_{new} \cdot V$, where V is the first value of each data segment. The prediction error is the difference between the predicted value, e.g., using linear or quadratic curve fitting, and the real value. When compressing data at EB_{base}, if the prediction error at EB_{new} is no greater than *HitRadius*, the data point is considered as a hit; otherwise, it is considered as a miss. Thus, scanning all data points when compressing data at EB_{base}, *HitRatio* at EB_{new} can be additionally obtained with essentially no extra overhead.

The results of *HitRatio* estimation are shown in Fig. 4, where the estimated *HitRatio* (in red) are compared against the real values (in blue) for error bounds from 10^{-9} to 10^{-1}. Overall the estimation of *HitRatio* is accurate, and the trend of *HitRatio* against error bound is well modeled. Nevertheless, the estimation deviates from the real value for *Dpot* at error bounds of 10^{-7} and 10^{-5}, and *Eddy* at 10^{-7}. We comment that the deviation is caused by the simplification in modeling *HitRatio*. Namely, during compression, SZ calculates *HitRadius* on the basis of segments that consists of 32 data points. However, during our estimation, we use the same *HitRadius* for all data points for simplification, which would otherwise require storing a vector of $HitRadius$ and can be expensive for large datasets. This simplification can cause inaccuracy for *HitRatio* prediction. Also, the prior work on SZ [7] suggests that one should use the preceding compressed values, instead of the original values to predict future values, so that the predicted values are bounded by the error bounds. In our case, since that the difference between compressed values and original values are limited by the error bound, which is typically under 10^{-1}, we use the preceding original values for prediction to simply the design.

NodeCount. As aforementioned, *NodeCount* can be estimated utilizing the distribution of quantization factors. We further notice that, among all datasets we evaluated, the distributions of quantization factors across different error bounds are highly similar, and they follow the Gaussian distribution. In general, to characterize a Gaussian distribution, one only needs to determine the mean and variance. A caveat is that, despite that quantization factor distributions of EB_{base} and EB_{new} are observed to have identical means, they exhibit different variances, as shown in Fig. 3. Therefore, when extrapolating *NodeCount* from EB_{base} to EB_{new}, the variance needs to be compensated. We notice that when the error bound is enlarged from EB_{base} to EB_{new}, assuming $EB_{new} > EB_{base}$, those data points missed under EB_{base} but hit under EB_{new} essentially extend the tails of the Gaussian

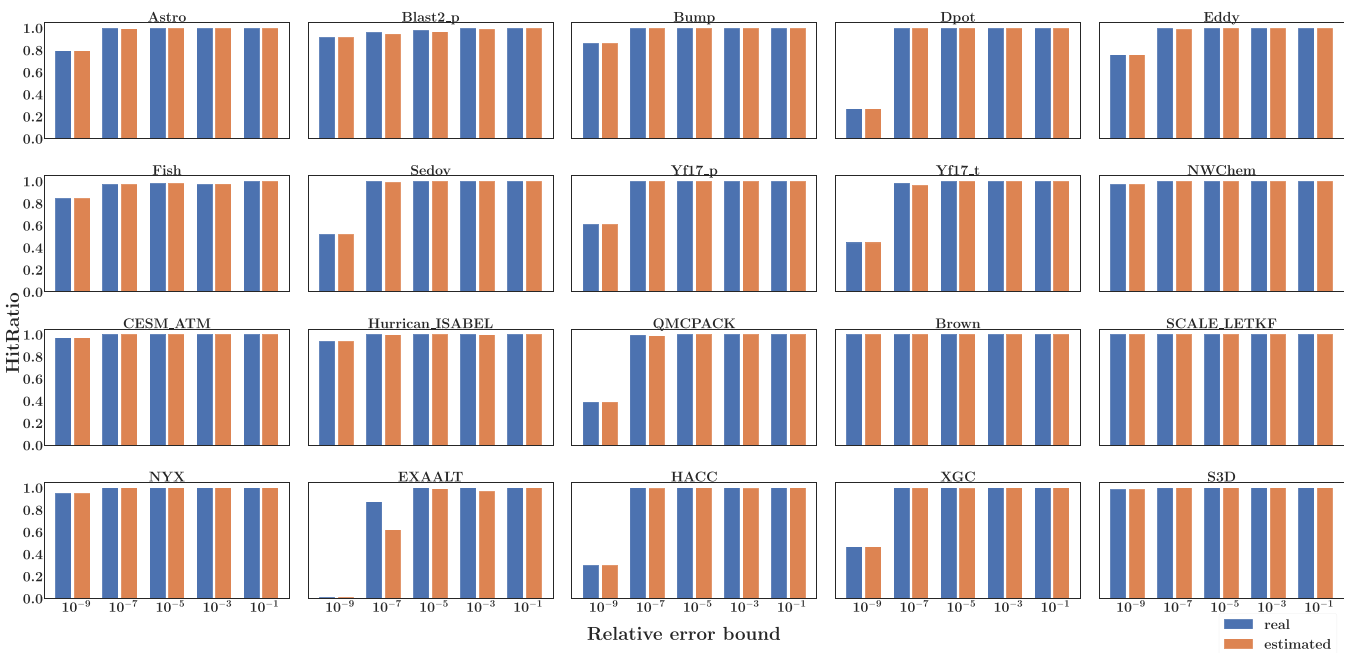


Fig. 4. *HitRatio* estimation on evaluation datasets. Note that EB_{base} is set to 10^{-9}.

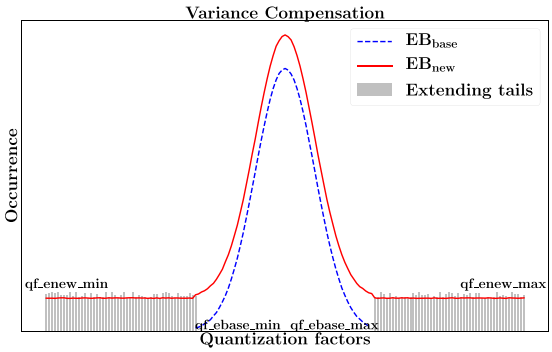


Fig. 5. Illustration of variance compensation for quantization factor distribution. In the figure, *qf_ebase_min* and *qf_ebase_max* stand for the smallest and largest quantization factor in the distribution of EB_{base} (blue curve). Meanwhile, *qf_enew_min*, and *qf_enew_max* stand for the smallest and largest quantization factor in the distribution of EB_{new} (red curve). Tails (grey bars) are added between *qf_ebase_min* and *qf_enew_min* as well as between *qf_ebase_max* and *qf_enew_max* to compensate the difference in variance.

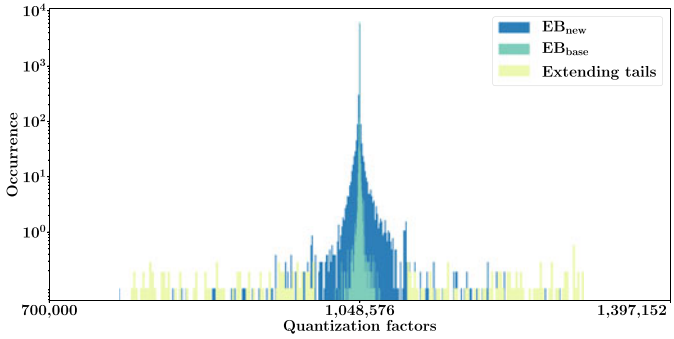


Fig. 6. Quantization factor variance compensation (*Astro*). The quantization factor at EB_{base} (in green) has a smaller variance, while the quantization factor at EB_{new} (in blue) has a larger variance due to the newly hit points on the tails. We use the uniform distribution to model the added tails (in yellow).

distribution under EB_{base}, thus increasing the variance. Hence, compensating the variance at EB_{new} comes down to obtaining the distribution of those newly hit points on both tails. Since the tails of Gaussian distribution are relatively smooth, we simplify the problem by using a uniform distribution to model the added tails.

We illustrate the variance compensation scheme in Fig. 5. The distribution of quantization factors at EB_{new} (red curve) has a wider range than the distribution at EB_{base} (blue curve). We model the added tails of newly hit data points (grey bars) using the uniform distribution. To this end, we need to identify the range of added tails, i.e., $[qf\_enew\_min, qf\_ebase\_min]$ and $[qf\_ebase\_max, qf\_enew\_max]$. We note that qf_ebase_min and qf_ebase_max are the minimum and maximum values of quantization factors at EB_{base}. For qf_enew_min and qf_enew_max, we use the minimum and maximum of newly hit data points as approximation. We note that the newly hit data points at EB_{new} can be easily obtained by simply comparing *HitRadius* with the curve-fitting prediction error.

Next, we apply the uniform distribution to randomly generate two sets of values in the range of $[qf\_enew\_min, qf\_ebase\_min]$ and $[qf\_ebase\_max, qf\_enew\_max]$, respectively, to compensate the difference in quantization factor distribution. The two sets of quantization factors generated from the uniform distribution are combined with the original distribution at EB_{base} to approximate the distribution at EB_{new}. We show the compensation results of *astro* in Fig. 6. We can see that the distributions under EB_{base} (in green bars) and EB_{new} (in blue bars) have the same mean value of 1,048,576. The variances of distribution under EB_{new} and EB_{base} are 37,446,718, and 375,682, respectively. And the compensated variance of distribution at EB_{base} with the extended tails is 35,443,720, which is very close to the true ilei1leivariance at EB_{new}. Once the quantization factor distribution at EB_{new} is obtained, we can estimate *NodeCount* by counting the number of unique quantization factors in the estimated distribution.

CompressionRatio. Based upon the *HitRatio* and *NodeCount* estimations, we next describe the complete *CompressionRatio* estimation. The process of *CompressionRatio* extrapolation from EB_{base} to EB_{new} involves the following steps:

Step 1: Run the standard SZ compression for a given dataset at EB_{base}. Calculate *HitRadius* at EB_{new}. Record the SZ compression metrics listed in Table 3. In addition, record the prediction error for each value.

Step 2: Extrapolate *HitRatio* to EB_{new} based on estimated *HitRadius* and the recorded prediction error.

Step 3: Calculate *qf_enew_min* and *qf_enew_max*. Construct the quantization factor distribution and estimate the *NodeCount* at EB_{new}.

Step 4: Estimate *TreeSize*, *EncodeSize*, and *OutlierSize*, based on the estimated *NodeCount* and *HitRatio*. The estimated size of compressed data, $OutputSize_{new}$, is the sum of the estimated $TreeSize_{new}$, $EncodeSize_{new}$ and $OutlierSize_{new}$.

In particular, for Step 4, the methodology used here, similar to our previous work [13], is based upon the following observations: 1) the Huffman tree size is proportional to the tree node count; 2) the encoding size is related to the depth of Huffman tree; and 3) the size of outliers is proportional to the number of outliers, with the size of each outlier being similar. The values of $TreeSize_{new}$, $EncodeSize_{new}$, and $OutlierSize_{new}$ at EB_{new} can be estimated as follows.

$$TreeSize_{new} = TreeSize_{base} \cdot \frac{NodeCount_{new}}{NodeCount_{base}}$$

$$EncodeSize_{new} = EncodeSize_{base} \cdot \frac{log_2 NodeCount_{new}}{log_2 NodeCount_{base}}$$

$$OutlierSize_{new} = OutlierSize_{base} \cdot \frac{OutlierCount_{new}}{OutlierCount_{base}}.$$

Therefore,

$$OutputSize_{new} = TreeSize_{new} + EncodeSize_{new} + OutlierSize_{new}$$

$$CompressionRatio_{new} = \frac{InputSize}{OutputSize_{new}}.$$

We take the *Astro* dataset as an example to illustrate the estimation of each component, as shown in Fig. 7. We can see that the two key factors, *HitRatio* and *NodeCount*, are well predicted. The non-linearity observed in *NodeCount* is also captured, which leads to the modeling of *TreeSize*, *EncodeSize*, and *OutlierSize*. More comprehensive evaluation of SZ *CompressionRatio* are presented in Section 5.

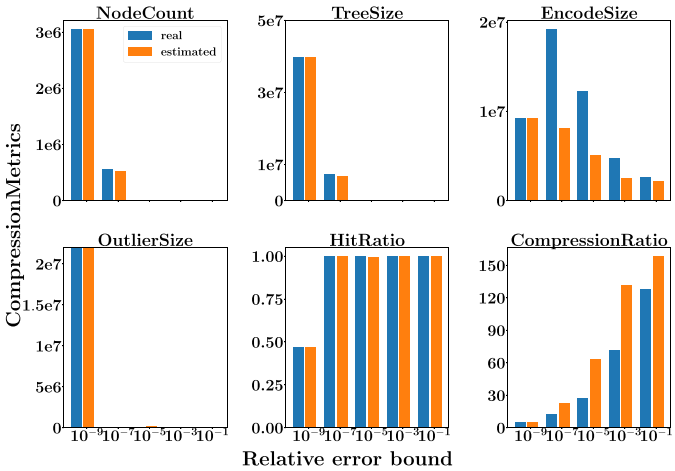


Fig. 7. SZ compression metric estimation.

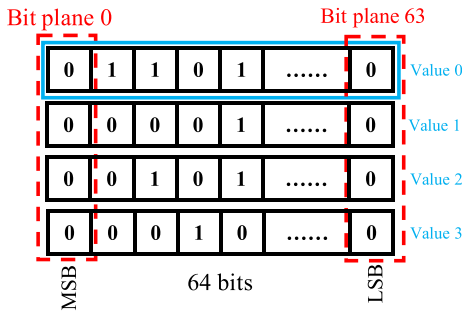


Fig. 8. Bit planes in a ZFP block. LSB and MSB represent the least and the most significant bit, respectively. ZFP encodes from bit plane 0 at MSB to bit plane 63 at LSB until the accuracy requirement is satisfied.

4.4 Prediction of ZFP Compression Ratio

4.4.1 ZFP Compression and Its Internal Metrics

ZFP compresses data based upon blocks. For each block, ZFP first transforms it into a set of floating-point mantissas along with a common exponent. The common exponent is computed from the largest absolute value in the block, resulting in mantissas in the range of $[-1, 1]$. Next, the floating-point mantissas are converted to fixed-point values and then taken into a reversible orthogonal transformation. The transformation, similar to the discrete cosine transform (DCT) used in JPEG image encoding, decorrelates spatially correlated values, resulting in near-zero transform coefficients that are typically more compressible. The transform incurs equal importance for each transform coefficient, and each bit of coefficients within the same bit plane has the same impact on accuracy [6], [22]. Thus, transform coefficients can be compressed using embedded encoding [23] where one bit plane of coefficients is encoded at a time. Note that the number of bit planes to be encoded can be calculated from the user specified precision.

ZFP can work in different modes depending on the user requirements. In this work, we choose the fixed-accuracy mode, in which the absolute error bound is set by the parameter *accuracy*. The data points in each block are compressed up to a common minimum bit planes to meet the target error tolerance.

MaxPrec and BitsPerBitplane. Given that ZFP compresses data by blocks, *OutputSize* is simply the sum of size of all compressed blocks (see in Equation (3)). We denote the compressed size of block i as $BlockSize_i$ (in bits), where $0 \leq i < n$ and n is the total number of blocks.

$$OutputSize = \frac{1}{8}\sum_{i=0}^{n-1} BlockSize_i. \tag{3}$$

In the extreme case where a block has all zero values, *BlockSize* is one and only a single bit of zero is written. If a block has non-zero values, *BlockSize* is the total number of bits used to encode the values in the block. To control the accuracy of compressed dataset, ZFP operates on one bit plane of coefficients at a time using embedded encoding (as shown in Fig. 8). When compressing the coefficients in a block by bit planes, *BlockSize* depends on *MaxPrec*, the number of bit planes to encode, and *BitsPerBitplane*, the number of bits consumed to encode each bit plane (see Equation (5)).

$$BlockSize_i = \sum_{j=0}^{MaxPrec-1} BitsPerBitplane_{ij}, \tag{4}$$

where $BitsPerBitplane_{ij}$ is the jth bit plane to encode for block i. In order to estimate *CompressionRatio* of ZFP, we must model and estimate these two parameters.

Note that ZFP in the fixed accuracy mode only supports the absolute error bound. In order to select a spectrum of error bounds that covers both loose and tight bounds, we set the absolute error bound to the product of root mean square (RMS) and the prescribed relative error bound. The definition of root mean square is defined as follows:

$$RMS = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}, \tag{5}$$

where $x_1, x_2, ...x_n$ is a set of values.

4.4.2 ZFP Modeling and Estimation

In this section, we aim to model and estimate two key compression metrics, *MaxPrec* and *BitsPerBitplane*, and further predict *CompressionRatio*. The central idea is to take advantage of the correlation between compression metrics and error bounds to make predictions.

As shown in Table 2, *MaxExp*, the maximum exponent value in a block, is constant across error bounds. *MaxPrec* decreases as the error bound loosens. It indicates that for each block, it requires fewer bit planes be encoded to satisfy the error tolerance. For *BitsPerBitplane*, although it decreases monotonically as well, we observe that the average number of bits to encode each bit plane decreases slowly from 3.50 to 3.37, when the error bound loosens from 10^{-9} to 10^{-3}. The reason is that this quantity highly depends on the actual bits of each bit plane but is not impacted by the error bound. Therefore, *BitsPerBitplane* is deemed to be less sensitive to the error bound, thus highly predictable at EB_{new}.

MaxPrec. We studied the ZFP implementation[1] and notice that *MaxPrec* is empirically calculated by Equation (6). In the equation, *MaxExp* is the largest exponent value in a block, and $log_2 Accuracy$ is the smallest bit plane number that should be encoded to [24].

1. This is of ZFP 0.5.3.

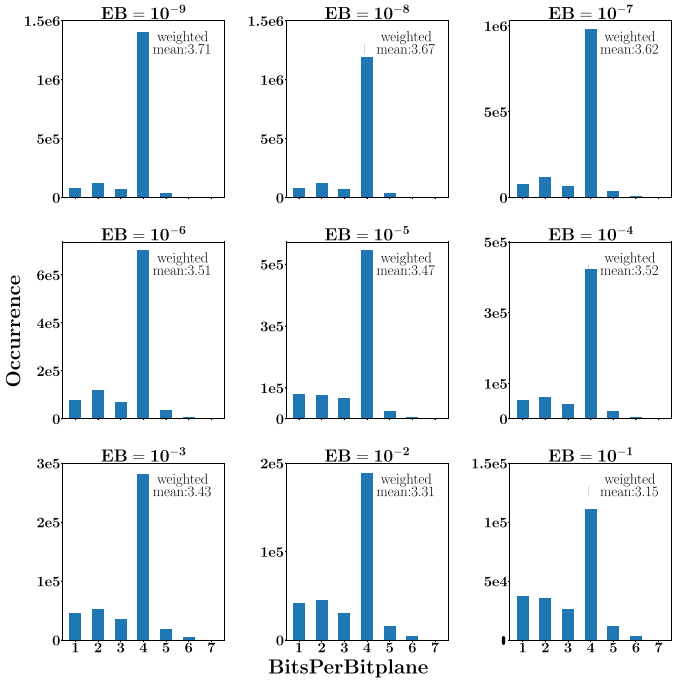


Fig. 9. Histogram of bits used in each bit plane (*Eddy*). Note that the x-axis is *BitsPerBitplane* and the y-axis is the occurrence of using the associated number of bits to encode. The distributions of *BitsPerBitplane* of other datasets are shown in Fig. 13 in Section 5.2.

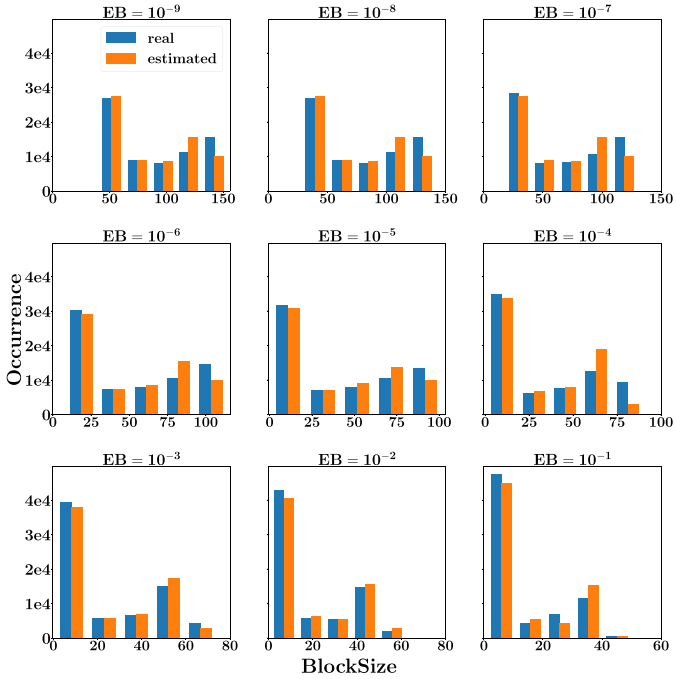


Fig. 10. Distribution of real and estimated *BlockSize* across error bounds (*Eddy*). The x-axis is the value of *BlockSize*, and y-axis is the number of blocks with the corresponding *BlockSize*.

$$MaxPrec = MaxExp - log_2 Accuracy + 2 \cdot (1 + d). \quad (6)$$

BitsPerBitplane. For a bit plane that has non-zero values, an embedded encoding scheme is applied to encode the bits of each of 4^d numbers into two parts: the first m bits are encoded verbatim where the value of m depends on the previous bit plane; then n bits are used to encode the remaining $4^d - m$ bits using run-length encoding. The total number of bits $(m + n)$ used to encode is in general data dependent, and it is non-trivial to calculate the exact number of bits used to encode each bit plane. Nevertheless, we observe that the distribution of *BitsPerBitplane* is highly similar for each dataset across error bounds. It is shown in Fig. 9 that for *Eddy*, each bit plane uses around 3.5 bits on average, and this stands true for all error bounds. Therefore, we use a weighted average of *BitsPerBitplane* at EB_{base} to approximate that at EB_{new}. In *eddy* for example, the total number of bit planes to encode at error bound 10^{-9} is 1715895, among which, there are $[78281, 120245, 68547, 1404963, 36905, 6882, 72]$ bit planes using $[1, 2, 3, 4, 5, 6, 7]$ bits to encode, respectively. Thus, the weighted average of *BitsPerBitplane* is 3.71.

The weighted averages of *BitsPerBitplane* for *Eddy* across error bounds are shown in Fig. 9. It is observed that *BitsPerBitplane* decreases slowly from 3.71 to 3.15 when the error bound loosens from 10^{-9} to 10^{-1}, validating the intuition that fewer bits are needed to encode each bit plane at a looser error bound. It also suggests that *BitsPerBitplane* is insensitive to error bound. This conclusion stands for all twenty datasets and the complete results of *BitsPerBitplane* distributions are shown in Fig. 13 (Section 5.2).

CompressionRatio. Now that we have modeled *MaxPrec* and *BitsPerBitplane*, $BlockSize_i$ can be approximated by the product of *MaxPrec* and the weighted average of $BitsPerBitplane_{ij}$, denoted as $\overline{BitsPerBitplane_i}$.

$$\begin{aligned} BlockSize_i &= \sum_{j=0}^{MaxPrec-1} BitsPerBitplane_{ij} \\ &\approx MaxPrec \cdot \overline{BitsPerBitplane_i}. \end{aligned} \quad (7)$$

We take the *Eddy* dataset as an example to show the estimation result of *BlockSize*. As shown in Fig. 10, the distributions of real (in red) and estimated (in blue) *BlockSize* are highly similar across error bounds. We note that the estimation error is a result of using weighted average of *BitsPerBitplane* at EB_{base} for prediction. With *BlockSize* modeled, *OutputSize* can be estimated using Equation (3). The process of compression ratio estimation can be broken down into the following steps:

- *Step 1*: For a given dataset, we run the ZFP compression in the fixed-accuracy mode at EB_{base}, to obtain *MaxExp* of each block and *BitsPerBitplane* of each bit plane.
- *Step 2*: We calculate the weighted average of *BitsPerBitplane* at EB_{base} and calculate the *MaxPrec* for error bound EB_{new}.
- *Step 3*: We estimate *BlockSize* at EB_{new} using Equation (7) and calculate the compressed data size *OutputSize* and *CompressionRatio*.

The results of compression ratio prediction for all datasets can be found in Section 5.2.

5 Evaluation

In this section, we present evaluations of our models for SZ and ZFP, respectively, on twenty real scientific datasets.

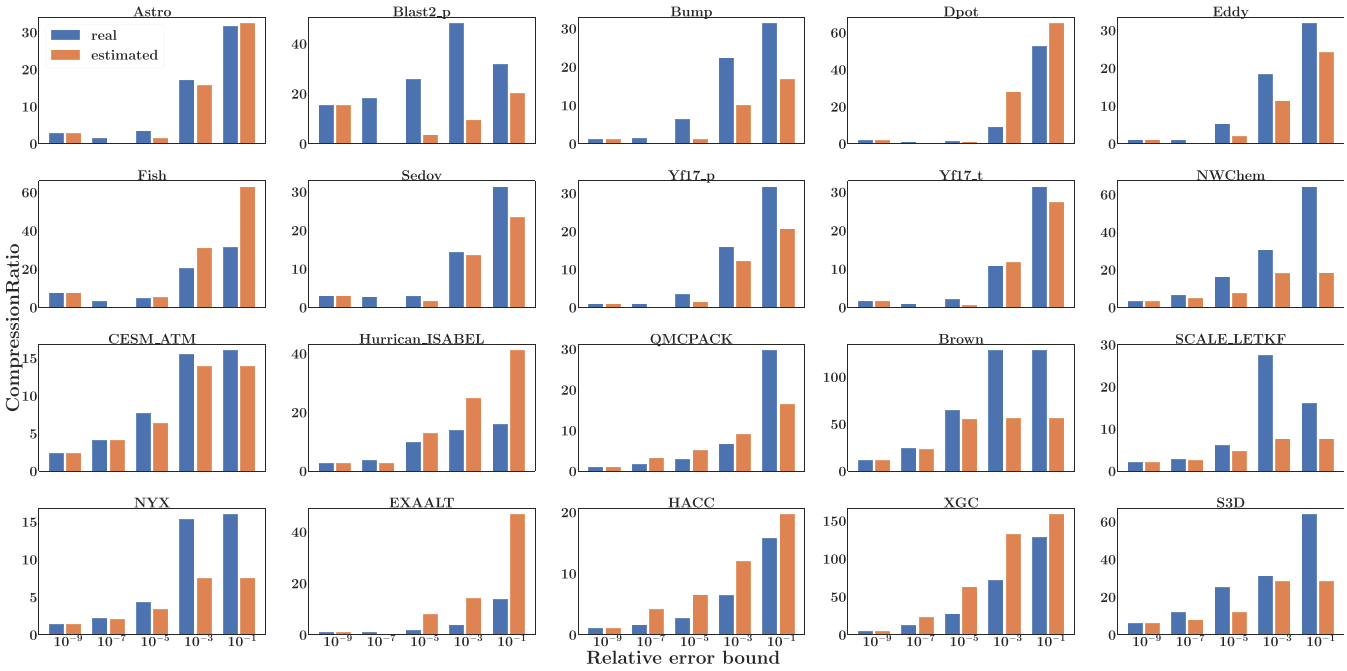


Fig. 11. SZ compression estimation.

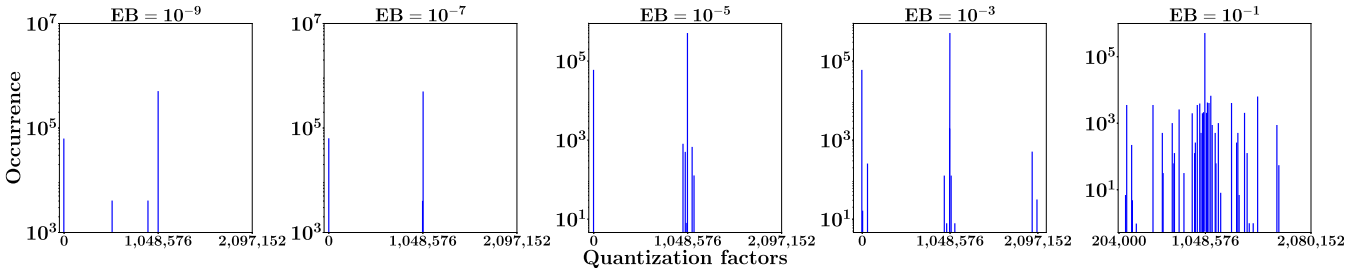


Fig. 12. Quantization factor distribution (*Blast2_p*).

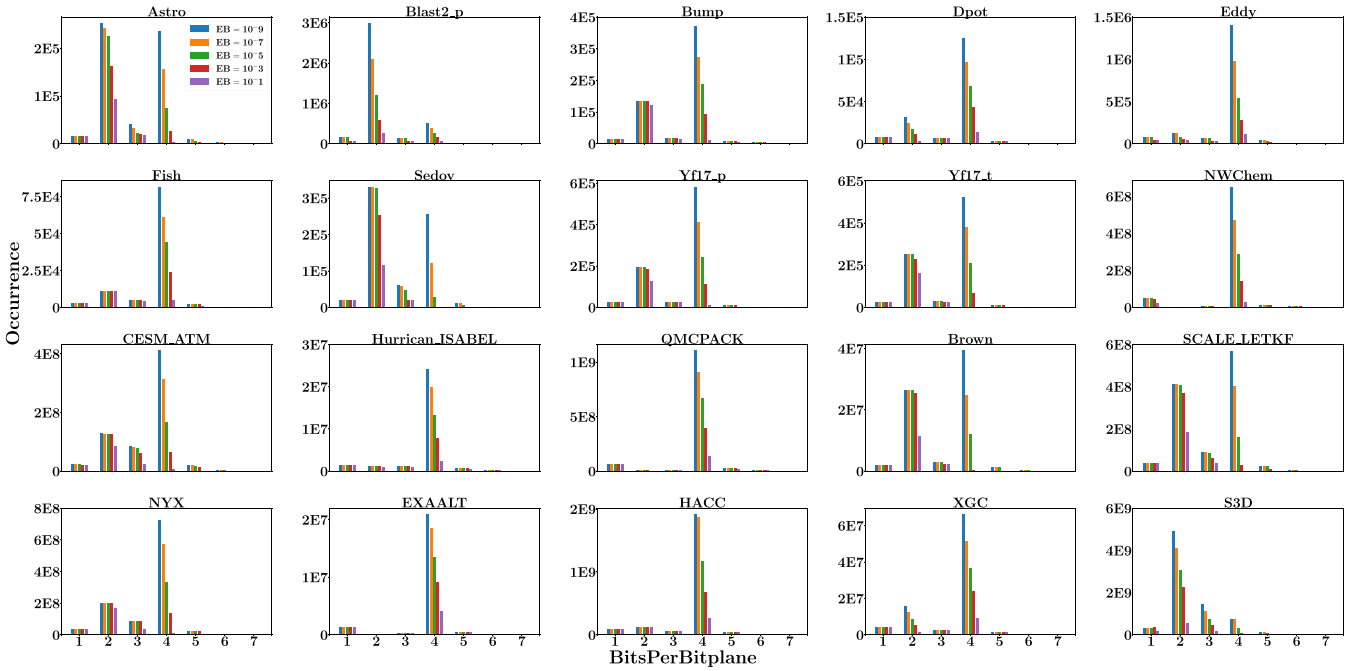


Fig. 13. Distribution of *BitsPerBitplane* over error bound. Note that the y-axis label for each plot is occurrence of bit planes. We observe that in each dataset, the distributions of *BitsPerBitplane* at different error bounds (bars with different colors) are highly similar.

TABLE 5
Weighted Average of $BitperBitplane$

Dataset	Weighted average
Astro	3.50
Blast2_p	2.95
Bump	2.28
Dpot	3.67
Eddy	3.71
Fish	3.31
Sedov	3.44
Yf17_p	3.58
Yf17_t	2.88
CESM_ATM	3.41
EXALLT	3.85
Hurricane_ISABEL	3.76
HACC	3.77
NYX	3.48
NWChem	3.82
QMCPACK	3.85
S3D	3.01
XGC	3.52
Brown	3.18
SCALE_LETKF	3.13

Among them, eleven datasets are adopted from a suite of *Scientific Data Reduction Benchmarks* [25] which contains data from real-world scientific simulations, including climate simulation [26], hurricane simulation [27], cosmological simulation [28], molecular dynamic simulation [29], N-body cosmological simulation [30], example molecular 2-electron integral values [31], weather simulation [32], many-body ab initio Quantum Monte Carlo [33], combustion simulation [34], and fusion simulation [35]. We compare the estimated compression ratios with the real ones across error bounds. Note that EB_{base} is set to 10^{-9}. We also quantitatively analyze the estimation error of compression ratios for both compressors.

5.1 SZ Compression Estimation

We apply the proposed estimation scheme to extrapolate the compression ratios of SZ across error bounds from 10^{-9} to other error bounds, as shown in Fig. 11. We further show the prediction error for each dataset in Fig. 15. For most datasets, the proposed scheme can capture the trend of compression ratio well and make reasonable estimation under most error bounds. However, for *Blast2_p*, the scheme shows a substantial departure from the real compression ratios. This is due to the very unique data distribution of *Blast2_p*, in which data points mostly center around two values. As a result, the quantization factor deviates from the exact Gaussian by a large margin (Fig. 12), and this in turn affects the accuracy of *NodeCount* estimation.

For *Eddy*, *Sedov* and *Yf17_p*, although the relative estimation errors shown in Fig. 15 are high, the absolute estimation errors are not. For example, for *Eddy*, the estimation errors at error bound 10^{-5} and 10^{-3} are 0.45 and 1.74, respectively. None of the absolute estimation errors exceeds an order of magnitude difference—the goal of compression estimation is to capture the trend of compression ratio, as opposed to predicting the precise value.

Furthermore, it is observed that for loose error bounds, e.g., 10^{-2} or 10^{-1}, the estimation is less accurate (except for *Astro*). The reduced accuracy is caused by the following: 1) our approximation of using uniform distribution to model newly hit points, and 2) the amount of newly hit points become very small as the error bound increases, thus making it hard to statistically capture their characteristics. We comment that loose error bounds are often not preferred in scientific productions due to the significant information loss, and they are listed here only for comparisons.

5.2 ZFP Compression Estimation

We first evaluate the estimation of *BitsPerBitplane*. In Fig. 13, we plot the distributions of *BitsPerBitplane* under error

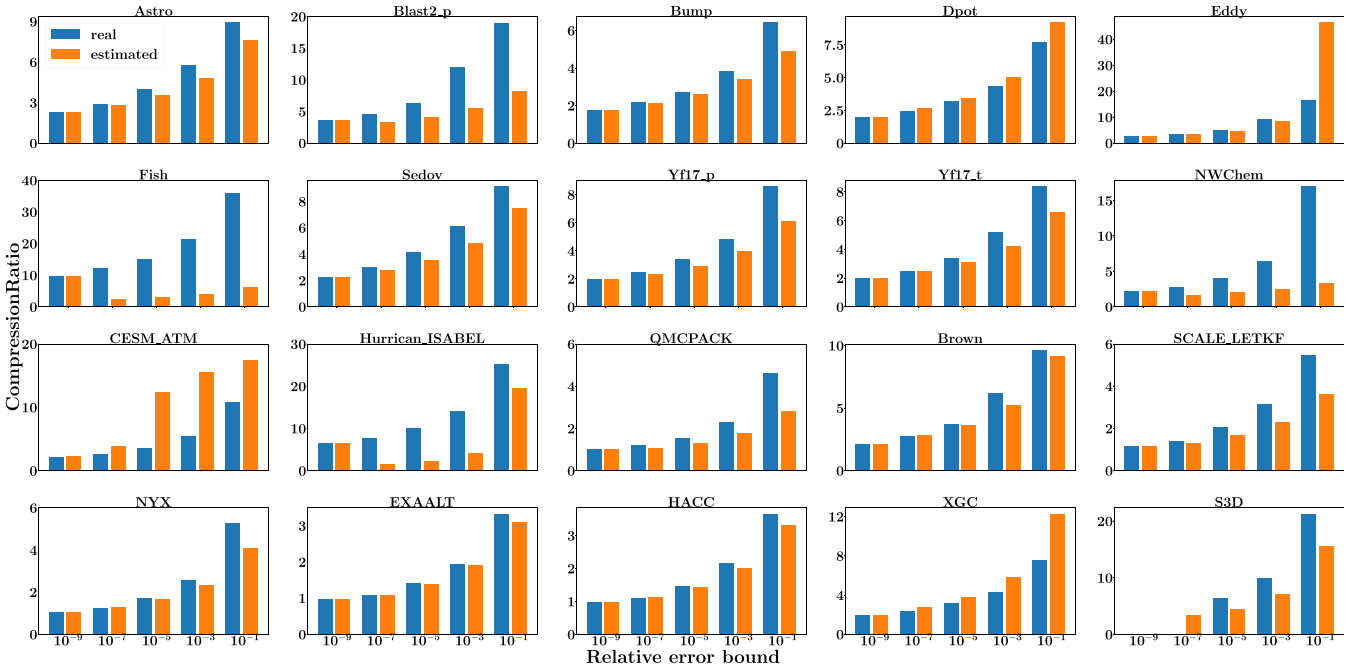


Fig. 14. ZFP compression estimation.

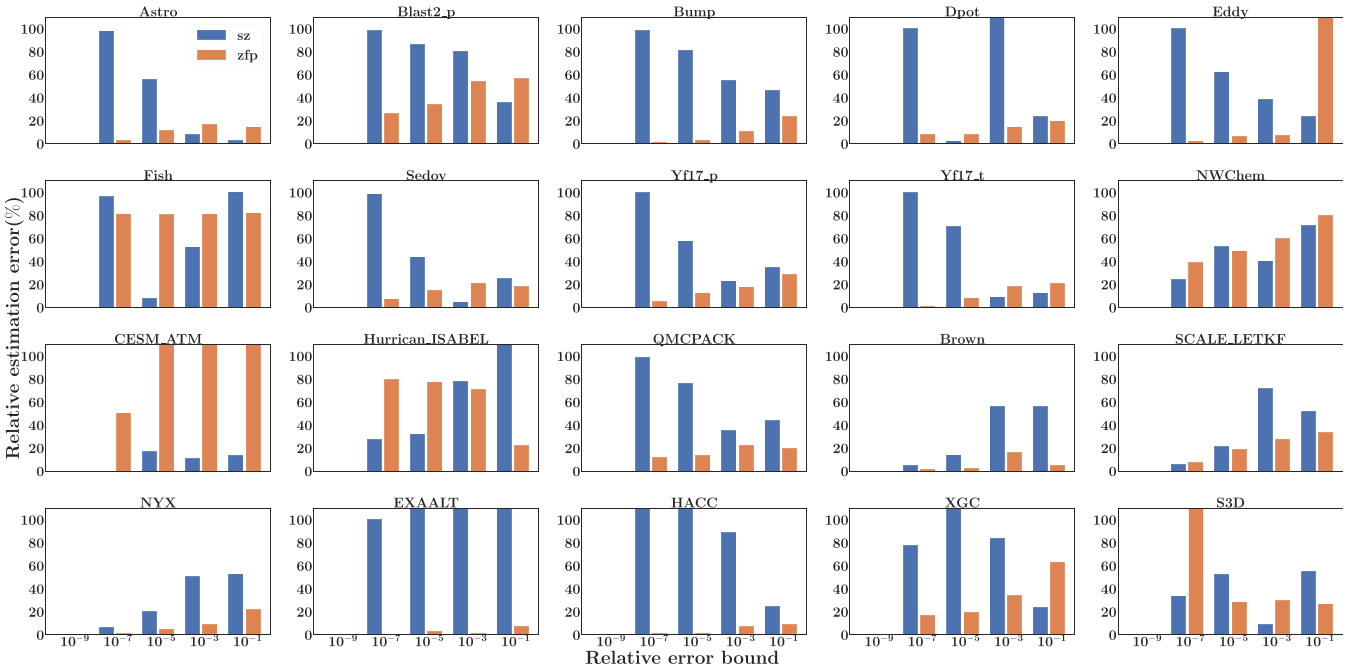


Fig. 15. Compression estimation error (SZ and ZFP).

bounds from 10^{-9} to 10^{-1}. It can be seen that the distribution of *BitsPerBitplane* maintains a similar shape across error bounds for all datasets. The weighted average of *BitsPerBitplane* under EB_{base} is listed in the Table 5.

The results of ZFP compression estimation are shown in Fig. 14. Overall, we can capture the trend of compression ratio well across error bounds. For *Yf17_p*, *Yf17_t* and *Sedov*, our proposed model over-estimates the output size which results in lower compression ratios. This is because *BitsPerBitplane* under $EB_{base}(10^{-9})$ is always larger than others (see in Fig. 13) due to tighter error tolerance. The relative estimation error of *CompressionRatio* for ZFP is shown in Fig. 15. As compared to SZ (Fig. 15), the estimation error of *CompressionRatio* for ZFP is significantly lower than SZ. For SZ, the quantization factor approximation involves significant simplifications of Gaussian tails, while ZFP does not have this problem. In addition, the estimation error generally increases as the error bound deviates from the EB_{base} (i.e., 10^{-9} in our runs). The largest relative estimation error observed is below 35 percent (for *Fish* and *Yf17_t* data at error bound 10^{-1}) while we observe that the absolute estimation error is 4.75 and 4.23, respectively.

6 Conclusions

Motivated by the insufficient understanding of lossy compressors, this paper thoroughly studies the mechanisms of two lossy compressors, SZ and ZFP. In particular, we examine how the error bound influences the compression ratio and identify the key factors that affect the outcome of compression. This work develops modeling techniques to predict the compression ratios based upon the estimation of key compression metrics across a set of error bounds. For SZ, we focus on the modeling and estimation of *HitRatio* and *NodeCount*; whereas for ZFP we capture the trend of *MaxPrec* and *BitsPerBitplane*. We evaluate the modeling and estimation schemes on real HPC datasets. The results show that our estimation scheme achieves a good accuracy on the compression ratios of SZ and ZFP for all datasets across error bounds. Our work is beneficial to domain scientists for choosing error bounds when compressing large datasets on HPC systems. A limitation of this work is that the SZ compression estimation is limited to the one-dimensional case. In the future, we plan to evaluate the case of multi-dimensional compression along with the performance prediction across compressors. In conjunction with this work, we plan to put together a complete estimation scheme for large-scale data compression.

Acknowledgments

The authors wish to acknowledge the support from the US NSF under Grant No. CCF-1718297, CCF-1812861, and NJIT research startup fund. The work performed at Temple University is partially supported by US NSF under Grant Nos. 1828363 and 1813081. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] M. Burtscher and P. Ratanaworabhan, "FPC: A high-speed compressor for double-precision floating-point data," *IEEE Trans. Comput.*, vol. 58, no. 1, pp. 18–31, Jan. 2009.

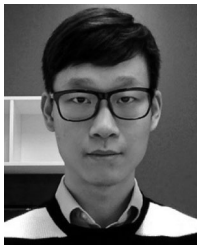
[2] D. Meister, J. Kaiser, A. Brinkmann, T. Cortes, M. Kuhn, and J. Kunkel, "A study on data deduplication in HPC storage systems," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2012, pp. 1–11.

[3] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 5, pp. 1245–1250, Sep. 2006.

[4] J.-l. Gailly, "GZIP: The data compression program," 2016. [Online]. Available: https://www.gnu.org/software/gzip/manual/gzip.pdf

[5] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. Samatova, "Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data," in *Proc. Eur. Conf. Parallel Process.*, 2011, pp. 366–379.

[6] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 12, pp. 2674–2683, Dec. 2014.
[7] S. Di and F. Cappello, "Fast error-bounded lossy HPC data compression with SZ," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2016, pp. 730–739.
[8] W. Austin, G. Ballard, and T. G. Kolda, "Parallel tensor compression for large-scale scientific data," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2016, pp. 912–922.
[9] M. Deering, "Geometry compression," in *Proc. 22nd Annu. Conf. Comput. Graph. Interactive Techn.*, 1995, pp. 13–20.
[10] D. Cohen-Or, D. Levin, and O. Remez, "Progressive compression of arbitrary triangular meshes," in *Proc. Conf. Vis. Celebrating Ten Years*, 1999, pp. 67–72.
[11] T. Lu, E. Suchyta, D. Pugmire, J. Choi, S. Klasky, Q. Liu, N. Podhorszki, M. Ainsworth, and M. Wolf, "Canopus: A paradigm shift towards elastic extreme-scale data analytics on HPC storage," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2017, pp. 58–69.
[12] A. Pukhov, "Particle-in-cell codes for plasma-based particle acceleration," in, *Proceedings of the CAS-CERN Accelerator School: Plasma Wake Acceleration*. Geneva, Switzerland: CERN, 2014.
[13] 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 *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2018, pp. 1–10.
[14] S. Wold, "Spline functions in data analysis," *Technometrics*, vol. 16, no. 1, pp. 1–11, 1974.
[15] X. He and P. Shi, "Monotone b-spline smoothing," *J. Amer. Statistical Assoc.*, vol. 93, no. 442, pp. 643–650, 1998.
[16] 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 *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2017, pp. 1129–1139.
[17] 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 *Proc. IEEE Int. Conf. Big Data*, 2018, pp. 438–447.
[18] A. M. Gok, S. Di, Y. Alexeev, D. Tao, V. Mironov, X. Liang, and F. Cappello, "PaSTRI: Error-bounded lossy compression for two-electron integrals in quantum chemistry," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2018, pp. 1–11.
[19] R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola, "Tthresh: Tensor compression for multidimensional visual data," *IEEE Trans. Vis. Comput. Graphics*, 2019.
[20] D. Tao, S. Di, X. Liang, Z. Chen, and F. Cappello, "Optimizing lossy compression rate-distortion from automatic online selection between SZ and SFP," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 8, pp. 1857–1871, Aug. 2019.
[21] D. Harnik, R. I. Kat, O. Margalit, D. Sotnikov, and A. Traeger, "To zip or not to zip: Effective resource usage for real-time compression," in *Proc. USENIX Conf. File Storage Technol.*, 2013, pp. 229–242.
[22] P. Lindstrom, P. Chen, and E.-J. Lee, "Reducing disk storage of full-3D seismic waveform tomography (F3DT) through lossy online compression," *Comput. Geosciences*, vol. 93, pp. 45–54, 2016.
[23] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Process.*, vol. 41, no. 12, pp. 3445–3462, Dec. 1993.
[24] P. Lindstrom, "ZFP 0.5.4 documentation," 2016. [Online]. Available: https://zfp.readthedocs.io/en/release0.5.4/modes.html#fixed-accuracy-mode
[25] C. Franck *et al.*, "Scientific data reduction benchmarks," 2018. [Online]. Available: https://sdrbench.github.io/
[26] "Community earth simulation model (cesm)," 2018. [Online]. Available: http://www.cesm.ucar.edu/
[27] "A simulation of a hurricane from the national center for atmospheric research," 2018. [Online]. Available: http://vis.computer.org/vis2004contest/data.html
[28] "Hardware/hybrid accelerated cosmology code (hacc)," 2018. [Online]. Available: https://press3.mcs.anl.gov/cpac/projects/hacc/
[29] "Exaalt: Molecular dynamics at exascale for materials science," 2018. [Online]. Available: https://www.exascaleproject.org/project/exaalt-moleculardynamics-at-the-exascale-materials-science/
[30] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Lukić, and E. Van Andel, "Nyx: A massively parallel AMR code for computational cosmology," *Astrophysical J.*, vol. 765, no. 1, 2013, Art. no. 39.
[31] J. Fermann and E. Valeev, "LIBINT: Machine-generated library for efficient evaluation of molecular integrals over gaussians," 2013. [Online]. Available: http://libint. valeyev. net/
[32] G.-Y. Lien, T. Miyoshi, S. Nishizawa, R. Yoshida, H. Yashiro, S. A. Adachi, T. Yamaura, and H. Tomita, "The near-real-time SCALE-LETKF system: A case of the september 2015 kanto-tohoku heavy rainfall," *Sci. Online Lett. Atmosphere*, vol. 13, pp. 1–6, 2017.
[33] J. Kim, A. D. Baczewski, T. D. Beaudet, A. Benali, M. C. Bennett, M. A. Berrill, N. S. Blunt, E. J. L. Borda, M. Casula, D. M. Ceperley, *et al.*, "QMCPACK: An Open source ab initio quantum monte carlo package for the electronic structure of atoms, molecules and solids," *J. Physics: Condensed Matter*, vol. 30, no. 19, 2018, Art. no. 195901.
[34] H. Kolla and J. H. Chen, "Turbulent combustion simulations with high-performance computing," *Modeling Simul. Turbulent Combustion*, Springer, pp. 73–97, 2018.
[35] C. Chang, "Multiphysics magnetic fusion reactor simulator, from hot core to cold wall," 2018. [Online]. Available: https://www.olcf.ornl.gov/caar/xgc/



Jinzhen Wang received the BS degree from Shandong University, China, in 2015, and the MS degree in electrical engineering from the New Jersey Institute of Technology, in 2017. He is currently working toward the PhD degree in the Department of Electrical and Computer Engineering at NJIT. His research interests include High Performance Comptuting and Cloud Computing.

Tong Liu received the BS degrees in computer science from the Huazhong University of Science and Technology, China, in 2015. He is currently working toward the PhD degree at Temple University. His research interests include computer systems, data storage, cloud computing, high performance computing, and data reliability.

Qing Liu received the BS and MS degrees from the Nanjing University of Posts and Telecom, China, in 2001 and 2004, respectively, and the PhD degree in computer engineering from the University of New Mexico, in 2008. He is an assistant professor with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology and Joint Faculty with Oak Ridge National Laboratory. Prior to that, he was a staff scientist at Computer Science and Mathematics Division, Oak Ridge National Laboratory for seven years.

Xubin He received the BS and MS degrees in computer science from the Huazhong University of Science and Technology, China, in 1995 and 1997, respectively, and the PhD degree in electrical engineering from the University of Rhode Island, Kingston, RI, in 2002. He is currently a professor with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA. His research interests include computer architecture, data storage systems, virtualization, and high availability computing. He received the Ralph E. Powe Junior Faculty Enhancement Award in 2004 and the Sigma Xi Research Award (TTU Chapter) in 2005 and 2010. He is a senior member of the IEEE, a member of the IEEE Computer Society and USENIX.

Huizhang Luo received the BS and PhD degrees in computer science from Chongqing University, China, in 2012 and 2017, respectively. He is currently a postdoctoral researcher with the Department of Electrical and Computer Engineering at NJIT. His research interests include memory systems, high performance computing, and nonvolatile memory.

Weiming He received the BS degree in physics from Shanghai Jiao Tong University, China, in 2016. He is currently working toward the MS degree in the Department of Electrical and Computer Engineering, New Jersey Institute of Technology.