Ocelot: An Interactive, Efficient Distributed Compression-as-a-Service Platform with Optimized Data Compression Techniques

Yuanjian Liu, Sheng Di*, *Senior Member, IEEE*, Jiajun Huang, Zhaorui Zhang, Kyle Chard, Ian Foster, *Fellow, IEEE*

Abstract—Large volumes of data generated by scientific simulations, genome sequencing, and other applications need to be moved among clusters for data collection/analysis. Data compression techniques have effectively reduced data storage and transfer costs. However, users' requirements on interactively controlling both data quality and compression ratios are non-trivial to fulfill. We propose a novel Compression-as-a-Service (CaaS) platform called *Ocelot* with four important contributions: (1) It offers real-time visualization, interactive compression, and transfer of scientific datasets. (2) It incorporates new strategies for compressing diverse types of datasets more effectively than traditional methods. (3) It provides an effective method for estimating the compression ratio and execution time of compression tasks. (4) Experiments on multiple real-world datasets on geographically distributed computers show that Ocelot can significantly improve data transfer efficiency with a performance gain of more than 10x in computing clusters with relatively slow networks.

Keywords—compression as a service (CaaS), data transfer, genome sequence compression, floating-point tensor compression

1 Introduction

Many scientific applications generate significant volumes of data that must be moved among geographically distributed sites. For example, particle accelerators such as Advanced Photon Source (APS) [1] generate data at up to 250 GB/s. Next-generation sequencing platforms such as DNBSQ-T20 can produce 22 TB of sequence per day. Molecular Dynamics (MD) simulations [2] that simulate 20 trillion particles in a long trajectory can generate 10 PB. Recent large language models such as Grok-1 grow to more than 380 GB. Most such data need to be moved to other computing clusters or personal computers for further analysis, model inference, or storage. Tools like Globus [3], [4], widely adopted on computing facilities (e.g., supercomputers) can accelerate transfer, but with such enormous datasets, transfer times can still be very long.

Many lossless and lossy compressors have been developed to address the big data issue. However, they cannot be employed directly or effectively for remote parallel data transfer tasks due to the following substantial drawbacks.

Limited/missing support for multi-node parallel compression. Off-the-shelf error-bounded lossy compressors lack efficient support for using processors on different compute nodes, which may significantly limit their parallel compression performance. On the one hand, the vast volume of data produced by many applications necessitates parallel compression across multiple nodes to complete the process within a reasonable time frame. On the other hand, the memory capacity of a single node is typically inadequate for most existing compression algorithms to handle some extremely large data files. To enable large file compression on multiple nodes, the algorithm should work with an efficient method that can split the files appropriately and coordinate processors on different compute nodes.

Unable to meet users' diverse requirements for compression quality. Although error-bounded lossy compressors have been effective in many applications, existing compressors are developed/driven based on relatively simple error control methods such as absolute error bound, inevitably leaving a significant gap in user requirements on the reconstructed data quality. Users often need to visualize the reconstructed data to determine whether they are valid in practice. Error-bounded lossy compressors, however, may introduce undesired artifacts in the reconstructed data, which is a non-trivial issue as the artifacts are related to many factors such as datasets, compressor design, error-bound types, and values. As such, enabling users to interactively check the quality of compression in real time and adjust compression parameters timely is a substantial feature to guarantee user requirements on compression quality.

Limited/missing support for diverse types of datasets/files. In the data repositories, there are diverse types of scientific datasets. While the existing error-bounded compressors are suitable for many simulation datasets, they are not suitable for other types of data. For example, genome sequence data consist of text-based sequence identifiers (e.g. ATCGGC...), which cannot be well compressed by con-

* *Sheng Di (sdi@anl.gov) is the corresponding author.*

- *Yuanjian Liu and Kyle Chard are with the Department of Computer Science at the University of Chicago, Chicago, IL 60601, USA.*
- *Sheng Di is with the Mathematics and Computer Science Division at Argonne National Laboratory, Lemont, IL 60439, USA.*
- *Jiajun Huang is with University of California - Riverside, Riverside, CA, 92521, USA.*
- *Zhaorui Zhang is with Department of Computing, The Hong Kong Polytechnic University, Kong Kong.*
- *Ian Foster is with both Argonne National Laboratory and the University of Chicago, Chicago, IL 60601, USA.*

ventional error-bounded lossy compression or binary-level lossless compression. Based on biological DNA similarity, dedicated reference-based compression algorithms can be much more efficient in this situation. Moreover, we note some inefficiency in the sequence alignment process and the compression of the quality score of existing genome sequence compressors such as Genozip [5]. On the other hand, existing error-bounded compressors such as SZ [6], [7] and ZFP [8] are not qualified for parallel (de)compression of very large files each of which is composed of one single tensor (or variable).

To address the above issues (also challenges), we develop a Compression-as-a-Service (CaaS) platform, *Ocelot* [1], for running optimized data compression/decompression/transfer tasks across multiple wide area network (WAN)-connected sites. Ocelot allows users to orchestrate remote tasks from their personal computers via a universal graphical interface. It supports multiple diverse lossy/lossless compressors including Gzip, Zstd, SZ3, ZFP, and our optimized genome data compressor. It also features an extensible module to support/plug more compressors easily on the CaaS platform.

The key contributions of this paper are as follows:

- We develop a CaaS framework, Ocelot, facilitated with Globus service and function-as-a-service (FaaS) techniques and with a universal graphical user interface for interactive control of compression/transfer among compute clusters.
- We propose a novel reference-based genome sequence compression algorithm with an improved sequence alignment technology and quality score compression method.
- We develop an efficient mechanism allowing users to interactively set multiple error bounds at different value ranges or regions for floating point tensors, which is critical to meeting diverse user requirements.
- We develop an efficient multi-node compression method with a layer-by-layer technique, to compress extremely large tensors/files that could not fit into memory, which is the first attempt to the best of our knowledge.
- We conduct experiments and benchmarks on multiple real-world datasets and demonstrate Ocelot's powerful capability to efficiently orchestrate data compression/transfer across heterogeneous supercomputers/clusters over WAN. Experiments show that the performance improvement exceeds 10x when transferring data from supercomputers to typical cloud computing clusters.

The rest of the paper is organized as follows. In Section 2, we discuss the research background and related work. In Section 3, we discuss our motivation and describe the design of our framework. In Section 4, we describe our proposed compression algorithms for genome sequence data and large floating-point tensors. In Section 5, we evaluate Ocelot on real-world scientific datasets, demonstrate the performance of our proposed compression algorithms, and show Ocelot's orchestration of compression/transfer among multiple clusters. Finally, we conclude the paper with a discussion of future work in Section 6.

1. The source code for Ocelot is available at https://github.com/legendPerceptor/Ocelot

2 Background and Related Work

Floating-point datasets/tensors are the major outputs of scientific applications and also represent the most substantial storage demand within deep learning models. Human genome sequences are also emerging as a common data type within computational facilities, with the prospect of affordable DNA sequencing for the masses on the horizon. This section explores the latest advancements in data storage and compression technologies for these data types, alongside recent initiatives to streamline the orchestration of remote tasks.

2.1 Error-bounded Lossy Compression

Error-bounded lossy compression techniques have emerged as indispensable tools for significantly reducing data volumes in floating-point tensors. These techniques play a crucial role in minimizing storage requirements, as evidenced by their applications in diverse domains such as molecular dynamics simulations [9], quantum computing [10], [11], and supercomputing environments [12]. By efficiently compressing data while ensuring that the error introduced during compression remains bounded, these methods can not only mitigate memory demands but can also alleviate I/O expenses in high-performance computing settings. Furthermore, they can eliminate the need for costly data recomputation [13]. Broadly speaking, error-bounded lossy compression models can be categorized into two main types: transform-based and prediction-based. The former involves applying transformations, such as wavelet transforms, to decorrelate raw data and subsequently employing specific encoders, such as embedded encoding techniques [8], to reduce coefficient data. On the other hand, prediction-based models utilize data predictors and linear-scale quantization to decorrelate datasets, followed by the application of variable-length encoding methods like Huffman encoding [14] and dictionary encoding such as LZ77 [15], to achieve high compression ratios. Prominent examples of prediction-based techniques include SZ [16], [17] and MGARDx [18].

2.2 Compression Performance Prediction

Knowing the expected compression ratio, quality, and time beforehand can be very beneficial for scientific workflows. Many previous works tried either white-box or black-box methods to predict the compression performance. The white-box methods usually do part of the compression and collect data from the compressor to predict the final compression performance. For example, Tao [19] developed a white-box method that samples data, estimates the probability density function of the data in the blocks and computes the entropy of the quantize values to derive a metric for compressibility. Jin' method [20] collects the distribution of quantization bins and uses a fixed formula with two tunable parameters to calculate the predicted compression ratio for SZ3. The existing white-box methods can be efficient but

cannot fit to all datasets and variations of the compressors. The black-box methods use data-centric features and predictors that are not derived from the internal mechanism of certain compressors. For instance, [21] extracts compressor-agnostic data features to determine the corresponding error bound for a target compression ratio. [22] and [23] compute statistical features derived from data including spatial diversity, spatial correlation, general distortion measurement, etc. to model the ease of compression on each dataset. Then they use the calculated features with a regression model to fit each compressor. The problem is that their selected features are quite expensive to compute. These methods are good in offline use cases but can be too heavy for optimizing the overall transfer time with compression. Therefore, we propose a method that combines compressor-related features, data-related features, and compressor configuration features for a fast and relatively accurate compression performance prediction for the SZ3 series of compressors.

2.3 Reference-based Sequence Compression

Raw sequencing data are typically stored in FASTQ format [24]. A FASTQ file consists of a separate entry for each short sequence, consisting of four lines: an identifier string, a nucleotide sequence (the read), the character '+', and quality scores. The identifier string contains information about the sequencing technology and other metadata obtained from the sequencing machine, which uniquely describes a read. The nucleotide sequence is a string of A, C, G, T, and N characters representing the bases (base-pairs) of the DNA sequence. The quality scores record the confidence of each base generated by the sequencing machine.

Existing FASTQ sequence compression algorithms can be categorized into two classes: reference-based and reference-free algorithms. *Reference-based algorithms* map the nucleotide sequences in a FASTQ file to a reference genome and use the mapped positions to encode the sequences. Examples include LW-FQZip [25], LW-FQZip2 [26], GTZ [27], and genozip [5]. *Reference-free algorithms* are used when a reference genome is not available. For example, Leon [28] and Quip [29] use assembly-based algorithms. Generally, reference-based compressors perform better in terms of both compression time and ratio than reference-free compressors, and thus we focus on developing a reference-based algorithm for FASTQ compression.

2.4 Remote Task Orchestration

The cloud-based Function-as-a-Service (FaaS) paradigm supports transparent remote function execution and data staging. The FaaS paradigm has been extended as a general model for remote computing across federated resources. Globus Compute [30] is one platform that uses FaaS as an interface to execute remote functions across the federated computing infrastructure. In the Globus Compute model, users can deploy endpoints on arbitrary computers. These endpoints are registered with the cloud-hosted Globus Compute platform and may then be used to execute functions. The cloud platform manages the secure and reliable execution of those functions on the selected endpoints.

Data transfer is essential in modern scientific computing. Globus Transfer is a widely used research data management platform that enables high-performance, secure, and reliable third-party data transfers. Globus Transfer builds upon the GridFTP protocol for data movement and adopts several optimization techniques such as parallel streams [31], [32], which can significantly improve data transfer performance. Transferring big data files with Globus Transfer, however, may still be time consuming due to limited network paths and underprovisioned data transfer nodes (DTNs). We aim to improve data transfer performance with Ocelot by applying dynamic compression methods.

3 Ocelot Framework Design

In this section, we describe our motivations and the design of the Ocelot framework.

3.1 Research Motivations and Goals

By consulting the users with real-world use-cases, we were highly motivated by several practical scenarios for developing the Ocelot Framework. (1) Some applications require proprietary software that is only available on a few clusters. (2) Some compressors or analysis programs depend on libraries that are optimized for specific architectures. For instance, the library "libdeflate" uses Intel Advanced Vector Extensions 512 (AVX-512) and may generate assembly instructions such as 'vpdpbusd' that are unavailable on relatively old AMD processors. For the above two reasons, data often have to be moved from one cluster to another for compression and/or analysis. (3) Data scientists hope to interactively compress/visualize the data so that they can check the quality of reconstructed data online thus selecting appropriate compression parameters in time instead of running the compression by a large batch compression script and checking the result after a while. (4) Authentication methods on different sites can be very different, causing users to need help from the support team every few months because they forget the exact authentication method for a specific site. Some sites support RSA authentication, while others require 2-factor authentication or a user-defined password plus a generated temporary passcode. Having a longer-term authenticated background program can be beneficial for many day-to-day tasks. We aim to address the above four problems with our Ocelot framework.

3.2 Ocelot Framework

We present an overview of Ocelot in Fig. 1, an orchestrator that interacts with the compressors, datasets, and transfer service on remote endpoints. The users interact with Ocelot through a graphical interface as shown in Fig. 2. We provide a fine-grained control over user-defined machines and allow users to choose different compressors against various datasets. Ocelot can map the datasets from multiple clusters to a logical directory so that users do not need to worry about the exact physical location of their desired datasets. We use color maps to offer a data preview option for floating-point tensors, which allow users to set multiple error bounds according to visual results. Users can also easily develop plugins to add their data analysis program to the Ocelot framework. With Ocelot, data scientists no longer need to repeatedly log in to each cluster to perform data

compression/analysis tasks and transfer the results back to their personal computers for visualization.

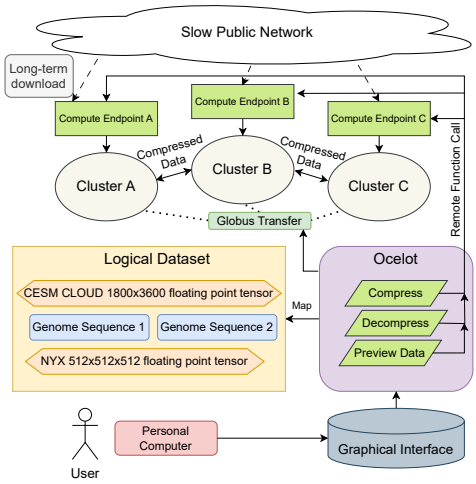

Fig. 1. Ocelot decouples the task execution from task manipulation and provides a universal data management interface for users to compress, transfer, analyze, and store various types of data.

Ocelot deploys long-term running endpoints on computing clusters, which gather information about datasets and compression performance. The information helps Ocelot predict the future compression time and determine the busyness of each cluster. The estimated compression time helps Ocelot decide the preallocated time for a batch job on shared systems. The benefit of this approach compared to the previous compression performance prediction work is that it does not have an additional cost when users perform a compression task. Moreover, the endpoint serves as a convenient stable downloader that can download datasets from the public network with a very slow transfer rate. The long-term running endpoint avoids the stable connection requirement to users' devices and thus can successfully download files for several days without encountering an SSH connection pipeline broken error or an accidental network drop that often occurs when users try to download datasets through their laptop.

Ocelot also has a mechanism to adapt to the availability of the compute nodes for transfer tasks that allow lossy compression. If the estimated compression/decompression time plus the transfer time of compressed files can benefit the overall throughput, Ocelot will run compression first. If a cluster cannot run the desired software, Ocelot can automatically perform a roundtrip approach: transfer the data to another cluster for compression and then transfer compressed data to the destination. We show in the following that due to high network bandwidths among supercomputers, the roundtrip approach can still reduce the total time for transferring data to cloud computing servers or personal computers which suffer slow networks. On supercomputers, data transfer is conducted on previously allocated Data Transfer Nodes (DTNs) and is managed through the Globus Transfer API. For personal computers and cloud computing servers, users can manually set up the endpoints and include them in Ocelot.

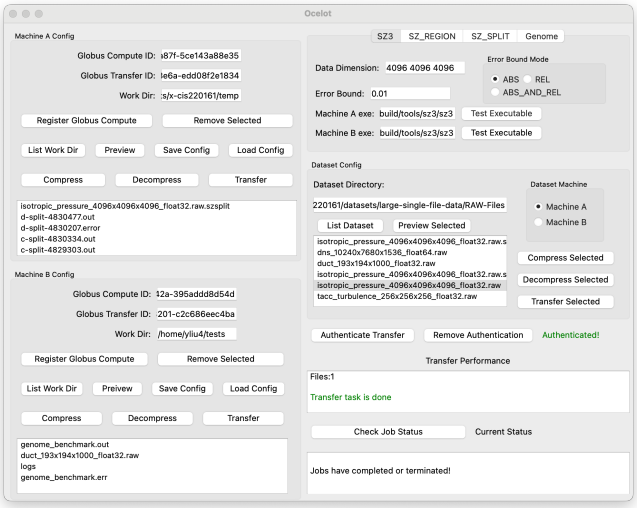

Fig. 2. Ocelot graphical user interface: users can control data transfer and compression with different algorithms between configured computing clusters. All the authentications are done when configuring the endpoints. Users no longer need to repeatedly authenticate when running jobs.

4 Diverse Compression Approaches: Genome Sequences and Floating-Point Tensors

In this section, we devise two optimized compression strategies under the Ocelot framework for two distinct types of data: genome sequence and floating-point tensors.

4.1 Genome Sequence Compression

We propose a novel reference-based genome sequence compression algorithm for FASTQ files. Our contribution mainly lies in an improved sequence alignment approach, lossy quality score compression, and Ocelot's remote orchestration capability for such compression.

For clarification, we first describe the reference-based genome sequence compression problem briefly. Genome sequence matching is a process of comparing the genetic information (DNA sequences) of different organisms to identify similarities or differences. The ideal scenario is that each read is just a subsequence of the reference (the exact match), so we only need to mark the matching position for each read. However, the sequence can have modification, insertion, and deletion that complicate the matching process. The existing algorithms often cannot match sequences with insertions and deletions well, resulting in a lower compression ratio. We aim to improve this alignment process.

We designed a parallel architecture for the genome sequence compression algorithm (shown in Fig. 3) because each read is strictly independent of others in a FASTQ file. The architecture employs a standard producer-consumer model, with one read thread, one write thread, and several worker threads to perform compression. These threads are synchronized by read and write buffers. This design allows FastqZip to compress extremely large FASTQ files without breaking memory limits and to achieve parallelism.

We employ a key-value map as an index for an efficient aligment process because naive long-string comparison is

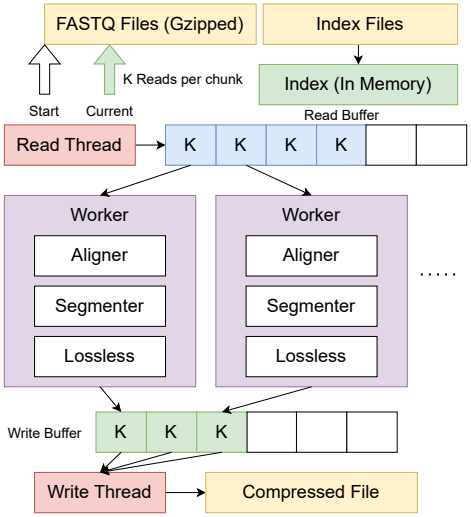

Fig. 3. Genome sequence compression architecture: The read thread must be sequential, but workers can proceed in parallel. The read buffer and write buffer allow maximum parallelism for the whole pipeline.

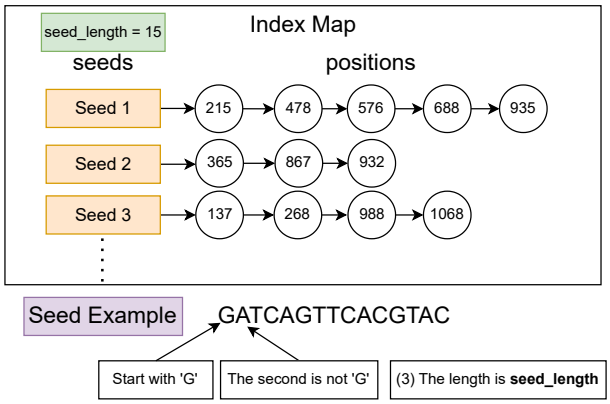

Fig. 4. Index concept: we look for all valid seeds in the reference sequence and record their positions. There are multiple positions because the same seed may appear multiple times in different locations on the reference sequence.

slow. We only need to build the index once for each reference sequence; once built, it can be loaded into memory rapidly during compression. As depicted in Fig. 4, the short seed sequences function as keys, and seed positions in the reference sequence serve as values. To simplify the storage of the index file, we propose three concepts: (1) forward sequence, (2) range index, and (3) forward index. The *forward sequence* connects the reference sequences to form one long sequence, and replaces all non-ACGT bases with 'A'. The *range index* is a fixed-length array used to store the cumulative number of repetitions for seeds, as shown in Fig. 5. The *forward index* stores the reference positions in seed-converted integer order. For example, in Fig. 5, seed1 and seed2 appear once each in the reference sequence, at positions 59 and 98, respectively, and seed3 appears three times, at positions 180, 340, and 790.

The core stage that improves the alignment capability in our algorithm is the alignment procedure, illustrated in Fig. 6. For each read, we iterate through the seeds in both forward and backward directions and calculate the starting position of the read in the reference sequence. If two seeds

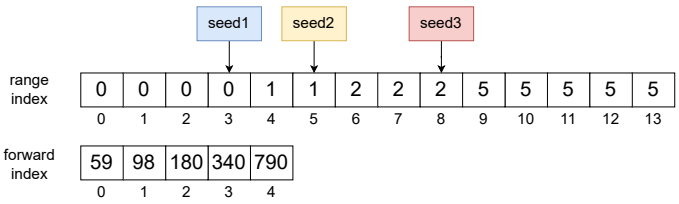

Fig. 5. Index storage: The range index and forward index arrays together store the reference positions for all seeds. A seed can be uniquely mapped to an index i in the range index array. The value in range index[i] is the starting index in the forward index array, and the value in range index[$i+1$] is the index after the ending index in the forward index array.

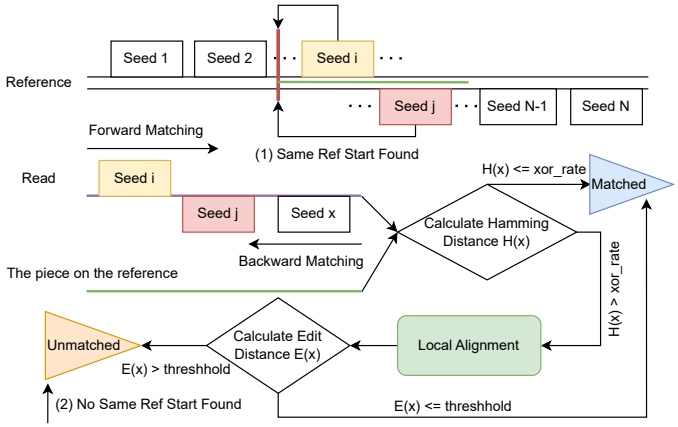

Fig. 6. Alignment procedure: when multiple seeds exist on a single read, if a match exists, two seeds should match to the same starting position on the reference. If the candidate sequence on the reference has a very low Hamming Distance against the read, it is a match. If there are the same starting positions, but the Hamming Distance is large, we use our proposed local alignment to find a match with insertion or deletion.

appear to have the same starting position, likely, the read is indeed cut from the reference at that position. We consider this read a matchable candidate when the same reference start is found. We need also to verify whether the match is exact. To make this process fast, we calculate the Hamming Distance [33], an XOR between two sequences. When there is no difference or only a few base modifications, the Hamming Distance will be small, and we can consider that a match is found.

We improve sequence alignment capability by further conducting a local search to calculate the Edit Distance [34] when the same reference start is found but the Hamming Distance is large. A matchable sequence with a large Hamming Distance is usually caused by insertion or deletion. Prior works [5], [35], [36] consider such cases unmatchable sequences. We use the WFA-2 algorithm [37], [38] to obtain the Edit Distance and the alignment CIGAR [39] to reconstruct the original read with insertions or deletions.

The segmentation process connects alignment results to form a single aggregated segment for better lossless compression. In sequence segmentation, we can further reduce the reference position storage by using the difference between positions (delta) when possible. Moreover, for paired FASTQ files, each alignment result stores two related reads $r1$ and $r2$, which usually form a reverse complement pair. We switch the forward read's result to $r1$ so there is a higher chance for the delta to be valid in the segmentation process. For quality segmentation, we propose a dominant bitmap solution, as illustrated in Fig. 7, to compress quality scores.

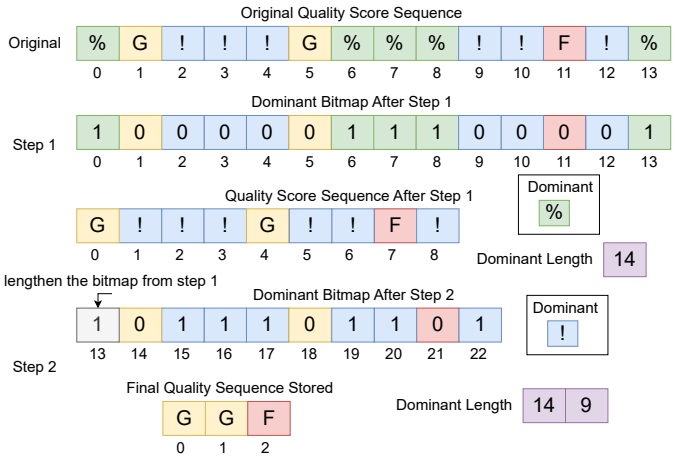

Fig. 7. Dominant quality bitmap generation: when a quality score is dominant over others, we use 1 to mark them and remove them in the quality score sequence. We continue to find a dominant quality score in the remaining sequence and repeat the process. In the end, only a few non-dominant qualities will remain in the sequence. We store a bitmap, a dominant length array, a dominant quality array, and the remaining quality sequence.

The idea is to use a bit instead of a byte to store each dominant quality and let the dominant quality be further compressed by a lossless compression algorithm such as the run-length algorithm. Moreover, we can cluster the scores to form fewer quality scores if the user allows a less fine-grained quality.

Many fields in our segmentation process can be further compressed by general-purpose lossless compression algorithms such as Zstd [40] and Zpaq [41]. The lossless compression mainly deals with repeated patterns such as a long sequence of 1s or 0s in our bitmaps. Since these compressors compress a stream of bytes, we consider them as a black box to reduce field sizes. It is worth noting that these compressors have to store some additional header information during compression and thus do not necessarily reduce the sizes for certain fields.

4.2 Innovations in Floating-point Tensors Compression

We identify two challenges in compressing floating-point tensors that have not been addressed in prior research. First, users may want to preview data rapidly before compressing and transferring an entire data file. Second, most current error-bounded compression methods, including SZ3, require loading the entire file into memory for compression. If the file's size exceeds memory limits, the compression process will crash with an out-of-memory issue.

To tackle the first problem, we developed a preview solution that involves a remote function call that instructs the computing endpoint to read the data and generate a visualization by converting floating-point numbers into colors. The preview aims to help users properly configure lossy compression without causing data to be unusable. To achieve a higher compression ratio, users may want to set multiple error bounds for a dataset to highlight the interesting value ranges or regions. However, since an error bound still has a gap to overall data quality such as visual quality, the error bounds set by users may still cause unexpected visual distortion in the data. One example is shown in Fig. 8 (A) and (B). The user knows that larger values in the dataset are more important so they set the error bound for [0.37, 0.83] to be 0.001, while keeping other value ranges with an error bound of 0.2. The compression ratio reaches 180, but the data are already severely distorted.

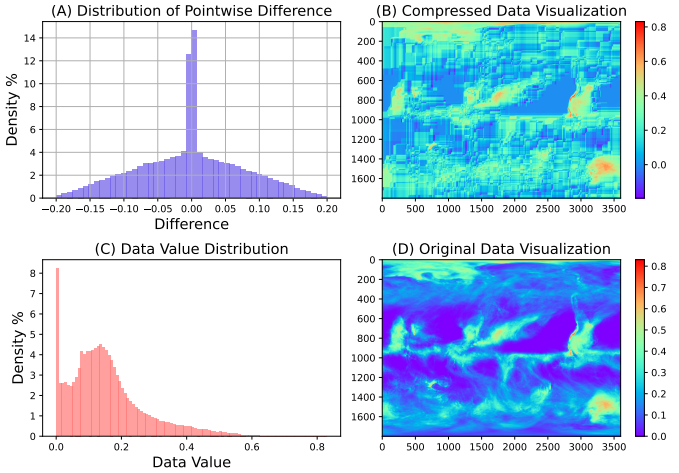

Fig. 8. CESM CLDMED multi-range compression distortion

The data preview mechanism provides users with both a visualization and a histogram of data value distribution ahead of the compression of full dataset: see Fig. 8 (C) and (D). Users can decide which value range to focus on by looking at the value distribution, and determine which regions to focus on by selecting rectangular regions on the visualized image. Setting the error bound in this way enables user to pay more attentions on data characteristics. For an extremely large file, users can select a layer to preview as shown in Fig. 9, which helps guide their region/range-based compression configuration. This preview does not involve compression and only transfers an image from the remote machine to the local machine. The compute and data transfer overhead is very low, and thus the preview is very responsive. Users can easily select a few layers from different parts of the data to have a good understanding of the data characteristics.

For the second challenge, we introduce a layer-by-layer compression technology[2] to compress exceedingly large files: see Fig. 10 (A). While data can be split in other ways, as shown in Fig. 10 (B), we favor layer-by-layer because (1) with other methods, the data points of a 'block' do not sit in continuous disk space and thus would require multiple seek operations that slow down I/O; and (2) the simplicity of layers makes it easy to parallelize the compression. A layer-by-layer streaming compression method transforms a 3D tensor into a sequence of 2D layers or slim 3D tensors. This technique can divide the files into smaller sections and then compress each section independently. It avoids out-of-memory errors when compressing large files.

To parallelize compression, we have two methods suitable for different computing environments: multi-threading and MPI programming. We first implement a multi-threaded parallel processing architecture similar to our

2. The C++ source code and Python binding is available at https://github.com/legendPerceptor/SZ3

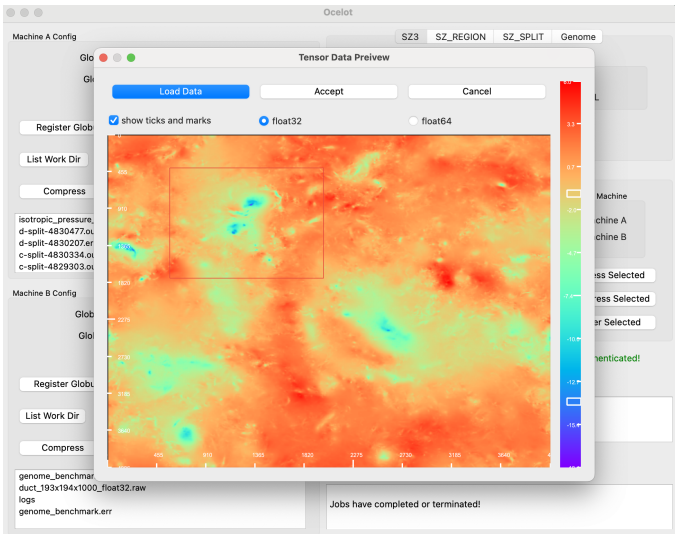

Fig. 9. The UI to preview one layer of the data in a large file. It also allows users to visually select different regions and ranges for compression settings.

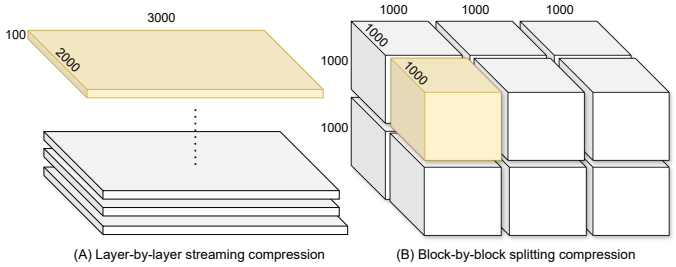

Fig. 10. Techniques to split large files into smaller blocks or layers to resolve the out-of-memory problem.

genome sequence compression algorithm, as depicted in Fig. 3. This architecture is suitable for cloud computing platforms that store data on a single or a small amount of SSD or HDD drives, where the I/O speed does not boost up with more I/O threads. For supercomputers, on the other hand, to scale up to multiple computing nodes and parallel file systems, we design a different architecture, as shown in Fig. 11. For compression, as the original file is huge, we can utilize multiple read processes to read the tensor in parallel. The read processes send each layer's information

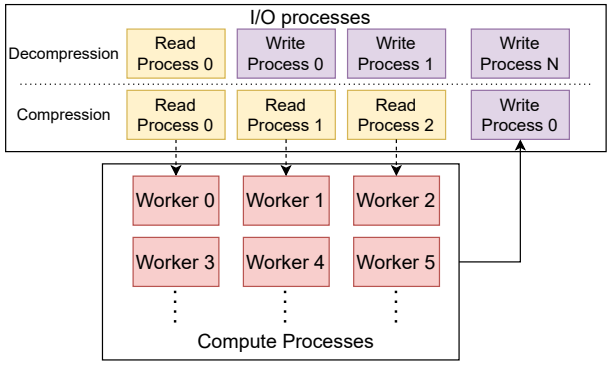

Fig. 11. Parallel layer-by-layer compression architecture for multiple processors on single/multiple compute nodes.

to preassigned worker processes for compression. We only use one write process because (1) the compression ratio of lossy floating-point tensor compression is usually quite high thus writing is much faster than reading and computing; (2) one writer allows the compressed chunks to be linearly connected without the need of an address book. For decompression, as the decompressed data are relatively large, we assign more processes to be writers. The advantage of this architecture allows processors on different nodes to compress a single file collectively. Moreover, parallel I/O is very suitable for our layer-by-layer compression method as each layer's offset can easily be calculated in advance.

4.3 Compression Performance Prediction

In this subsection, we propose a prediction model to estimate the lossy compression ratio, compression speed, and peak-to-noise ratio (PSNR) for prediction-based lossy compression algorithms including SZ and SZ3.

In general, users cannot predict compression quality (such as compression ratio and data distortion level) for a particular error-bounded lossy compressor without performing the compression on the given dataset. This is because the effect of data prediction/transformation and coding in the compressor varies with diverse data features. With our prediction model, users can quickly test multiple compression settings and choose the one that best matches their use case.

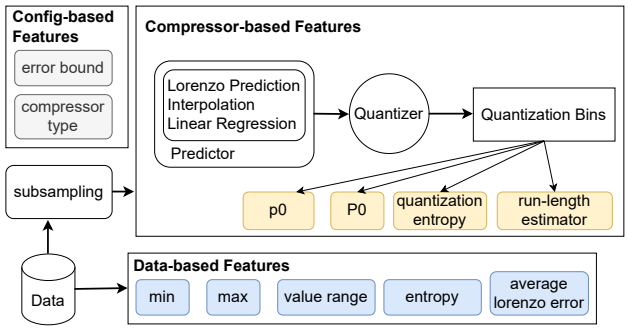

Fig. 12. The features used to predict compression quality are categorized into three types: config-based, compressor-based, and data-based features, which are shown as colored boxes.

We train a machine learning (ML) model on masses of sample datasets, with the aim to build a relationship between the compression-related features and the compression quality. The model can then be used to estimate compression quality accurately based on the features extracted from the given datasets at runtime.

We derive many features as input to our model, as illustrated in Fig. 12. Identifying a set of useful features is challenging, because (1) the extraction of each feature should have low computation cost, and (2) the features should form an accurate indicator of the compression quality. We consider features in one of three categories: (1) config-level features, (2) data-based features, and (3) compressor-level features.

Config-based features are configuration settings (including error bound values and compression pipeline) specified by users. Different error bounds can yield largely different

compression quality (e.g., compression ratios and compression speed). Compression quality also depends on specific compressors each with distinct designs. The prediction-based compressors [6], [7], for example, may adopt various predictors which may exhibit different performances. We enable our model to recognize the characteristics of compressors by treating the compressor-type feature as a discrete classification variable and feeding it with profiling data.

Data-based features describe the characteristics of datasets, which is also a key factor in distinguishing compressibility. As shown in TABLE 1, even for the same application, different datasets can have very different properties such as min, max, and value range. In addition, we also use byte-level information entropy as one feature, because it reflects the "chaos-level" of a dataset. The entropy is defined as

$$H(X) = -\sum_{x \in S} p(x) \log p(x) = E[-\log p(X)]$$

where S is the set of byte values (0-255) and p denotes the probability/frequency of an element in S. In general, the higher entropy a dataset exhibits, the more difficult it is to compress that dataset. As verified in Fig. 13 (a) and (b), the entropy value projects a positive correlation against the compression time, especially when the error bound is relatively low. It is worth noting that when the error bound is relatively high, the entropy would lose its effect (as shown in Fig. 13 (c)), because the large error bound would diminish the data variation. Moreover, we use the average Lorenzo error (i.e., the difference between the true data value and Lorenzo-predicted value [6]) as a feature to shape the "easiness of prediction" for a dataset. If the average Lorenzo error is high, the prediction stage tend to be imprecise, leading to low compression ratio.

TABLE 1
Examples of the basic data-based features in different datasets: CLDHGH, FLDSC, and PCONVT are three fields in the CESM dataset. HACC-VX and HACC-VY are two fields in the HACC dataset.

Dataset	CLDHGH	FLDSC	PCONVT	HACC-VX	HACC-XX
min	0.00	92.84	39025.27	-3846.21	0.00
max	0.92	418.24	103207.45	4031.25	256.00
value range	0.92	325.40	64182.18	7877.46	256.00

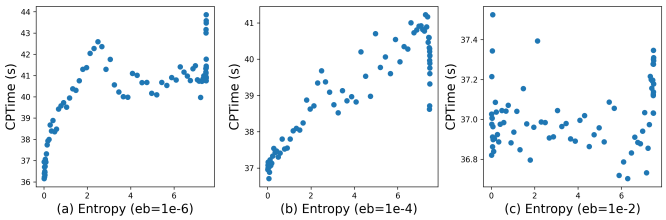

Fig. 13. Data entropy vs compression time in Reverse Time Migration (RTM) [42] application with three error bound settings

Compressor-based features are the properties of the intermediate data used in the course of lossy compression, which generally have the highest prediction ability for compression quality. Specifically, we focus on the quantization bins, as shown in Fig. 12. Since the quantization bins are encoded by the subsequent lossless encoders, its characteristic closely correlates to the final compression quality. In order to control the execution overhead, the quantization bins are computed based on the sampled data points. As demonstrated in Fig. 12, we develop four compressor-based features, including p_0, P_0, quantization entropy, and run-length estimator. (1) p_0 denotes the percentage of the 0-value bins over all quantization bins. In general, large p_0 tends to yield a high compression ratio and compression speed, because a large majority of predictions should be accurate in this situation. (2) P_0 denotes the fraction of '0'(encoded) taken in Huffman coding in the regard of the full Huffman encoded data size. (3) Quantization entropy is the entropy of quantization bins. If the prediction is accurate, quantization bin values will mostly be near 0, and the quantization entropy will be low. (4) Run-length estimator (denoted R_{rle}) is derived from P_0 and p_0 by the following equation: $R_{rle} = 1/((1-p_0)P_0 + (1-P_0))$.

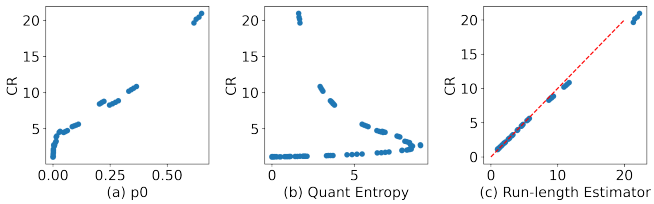

Fig. 14. The relationship between p_0, quantization entropy, run-length estimator and compression ratio for Nyx application.

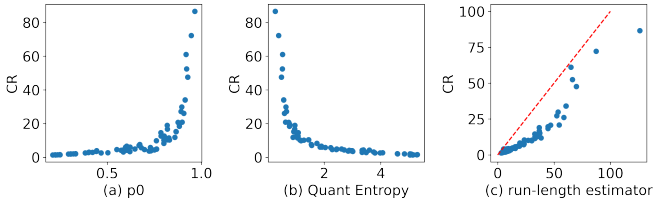

Fig. 15. Run-length estimator alone fails to predict the compression ratio for Miranda application while the three features together form a correlation to the compression ratio which can be learned by a machine-learning model.

Although the p_0 and P_0 are also used in related work [20], our solution is much more accurate in compression quality estimation in general cases. The estimation of compression ratio in [20] depends on the following formula: $\hat{CR} = 1/(C_1(1-p_0)P_0 + (1-P_0))$, where C_1 is an ad-hoc tuning parameter which varies with different applications. We use SZ3's default configuration as an example to show the relation between extracted features and the actual compression performance. As shown in Fig. 14 (c), almost all data points are located on the line $y = x$ (red line in the figure), which means the estimated compression ratio $\hat{CR}$ under that formula could be very accurate in this case. This is due to the fact that this formula happens to form a linear function with compression ratio for the Nyx [43] application. However, that formula is sensitive to the tuning of the C_1 parameter, which may cause unexpected large compression quality estimation errors in other applications. For instance, the estimator's value does not form a linear relationship with the compression ratio for the Miranda [44]

application (as shown in Fig. 15 (a) and (b)), which leads to bad compression quality estimation in turn (see Fig. 15 (c)). In comparison, our R_{rle} formula does not depend on the C_1. In fact, R_{rle} serves as a feature and we feed it into the ML model along with other features (including p_0 and P_0), and thus the model can automatically fine-tune the coefficients applied on those features, thus being able to keep an accurate estimation in most of cases.

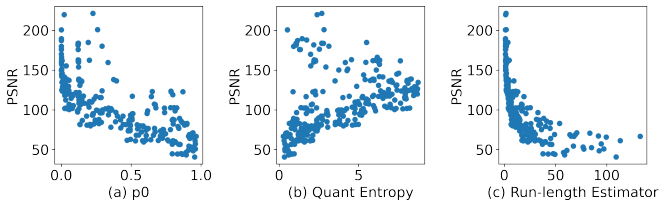

Fig. 16. CESM dataset — PSNR versus compressor-level features

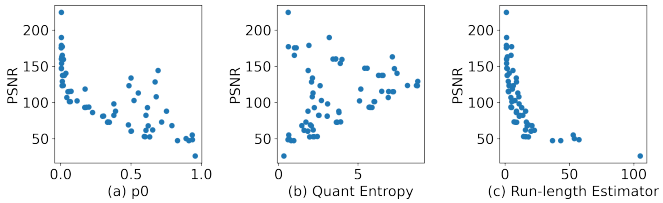

Fig. 17. ISABEL dataset — PSNR versus compressor-level features

Our compressor-based features can also be used to predict the reconstructed data distortion. This is because these features are also closely correlated to the data distortion metrics such as PSNR, as verified in Fig. 16 and Fig. 17.

5 Performance Evaluation

We present our experimental results on the proposed compression algorithms and the Ocelot framework. We first evaluate the performance of the genome sequence compression and compare it against other state-of-the-art compression algorithms. Then we evaluate the large 3D tensor compression and the effectiveness of our layer-by-layer compression techniques. Finally, we evaluate the compression run time estimation and end-to-end data transfer performance for the Ocelot framework.

5.1 Experimental Settings

We collect performance data on two ACCESS supercomputers (Purdue Anvil, Rockfish) and two Alibaba ECS machines, with specifications in Table 2. The network bandwidth for the two ACCESS endpoints are both 100 Gbps, while for Alibaba ECS, the default is 10 Mbps, with a maximum configuration of 1000 Mbps—highlighting a significant network disparity between supercomputers and cloud clusters.

We evaluate genome sequence compressors on three datasets sequenced on different platforms and with various lengths: see Table 3. (We refer to these datasets as A, B, C in the following.) For floating-point tensor compression, we employ 6 moderately sized datasets, QMCPACK [48], ISABEL [49], RTM, Miranda [44], CESM [50], and Nyx [43], and two large datasets, Forced Isotropic Turbulence and Turbulent Channel Flow [51]: see Table 4.

TABLE 2
Machine Specifications: Cores and Memory are the total amount in a single compute node.

Machine	CPU	Cores	Memory
Rockfish	Intel Xeon Gold Cascade Lake 6248R	64	192 GB
Purdue Anvil	AMD EPYC 7543 32-Core Processor	128	256 GB
Argonne Bebop	Intel Xeon E5-2695v4 & Phi 7230	36	128GB
Alibaba ecs.c7se.4xlarge	Intel Xeon Platinum 8369B	16	32 GB
Alibaba ecs.g7.32xlarge	Intel Xeon Platinum 8369B	128	512 GB

TABLE 3
Genome Sequence Datasets

Dataset	Platform	Symbol	Size
E100024251_L01_104 [45]	DNBSEQ-T7	A	18+20 GB
CL100076243_L01 [46]	BGISEQ-500	B	54+55 GB
E100030471QC960_L01 [47]	DNBSEQ-T7	C	28+27 GB

5.2 Genome Sequence Compression Evaluation

We compare the performance of our genome sequence compression algorithm against three modern genome sequence compression algorithms: Spring [52], GTZ [27], and genozip [5]. We measure compression ratio (CR), (de)compression CPU time, and (de)compression wall time. As shown in Fig. 18, Spring compresses more slowly and achieves a lower compression ratio than the other methods, and thus we do not consider it further. GTZ [27] is fast but has a lower compression ratio and it cannot successfully compress paired FASTQ file compression (probably due to an internal bug). Therefore, we use the single file compression benchmark to compare the four algorithms. We conclude that the best existing genome compression algorithm is Genozip [5] in terms of project completeness, ease of use, compression ratio, and compression speed. We present a more detailed

TABLE 4
Floating-point tensor datasets: the dimensions are layed out in the order of Z,Y,X, where X is the fastest changing axis in the memory.

Application	Description	Dimensions	Sizes
QMCPACK	electronic structure calculations of molecular, periodic 2D, and periodic 3D solid-state systems	33120×69×69 (float32)	150MB
ISABEL	temperature, speed, etc.	100×500×500 (float32)	95MB
RTM	seismic imaging in complicated areas	235×449×449 (float32)	180MB
Miranda	Hydrodynamics code for large turbulence simulations	256×384×384 (float32)	144MB
CESM	cloud, temperature, pressure in climate simulation.	1800×3600 (float32)	25MB
Nyx	density, temperature in cosmology simulation	512×512×512 (float32)	512MB
Turbulent Channel Flow	Pressure field of a direct numerical simulation of forced isotropic turbulence.	4096×4096×4096 (float32)	256GB
Forced Isotropic Turbulence	A pressure field of a direct numerical simulation of fully developed flow	1536×7680×10240 (float64)	900GB

comparison with Genozip in Table 5. Note that the compression ratio is calculated based on the gzipped original file sizes.

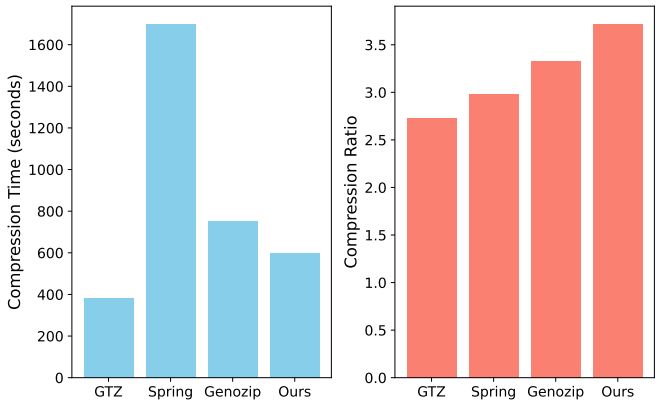

Fig. 18. Compression time and ratio comparison on the first file of E100024251_L01_104. Each algorithm uses 16 threads on the **ecs.c7se.4xlarge** machine.

TABLE 5
Our algorithm vs. Genozip. CR: compression ratio; CPTime: CPU time in compression; DPTime: CPU time in decompression.

Compressor	Dataset	CR	CPTime	DPTime
Our Algorithm	A	**3.37**	**151m48s**	**94m20s**
	B	**2.44**	**417m17s**	**251m15s**
	C	**2.54**	**522m16s**	**245m31s**
Genozip	A	3.14	160m28s	100m14s
	B	2.33	572m5s	303m54s
	C	2.45	526m43s	281m14s

Our findings indicate that our algorithm has demonstrated advantages in both compression ratio and compression time across the three paired sequence datasets. Our algorithm allocates additional time to sequence alignment to achieve a superior compression ratio. However, the overall (de)compression time is reduced due to our multi-threading architecture disregarding the order of reads and string identifiers. In contrast, Genozip ensures complete lossless compression. We assert the validity of our algorithm for the following facts: in FASTQ files, each read is self-contained, and the string identifier typically pertains only to sequencing machine specifications, which are inconsequential for downstream analysis.

We analyze our algorithm's memory and CPU utilization. In Fig. 19, memory usage ranges between 50% and 60% (∼19 GB) when all CPU resources are utilized. If memory is limited, say to 16 GB, reducing the number of threads or decreasing the read number can reduce the memory demand. On the other hand, our algorithm is capable of fully utilizing computing resources with an appropriate number of threads. Fig. 19 (B) shows that by setting the thread number to 16 (the number of available CPUs), each worker can take up one CPU core and reach over 90% of CPU utilization. The slight oversubscription shown in Fig. 19 (C) does not increase the CPU utilization more while having some troughs that drop to 65% of utilization. We can safely conclude that setting the thread number to the number of CPU cores available can utilize the computing resources well enough.

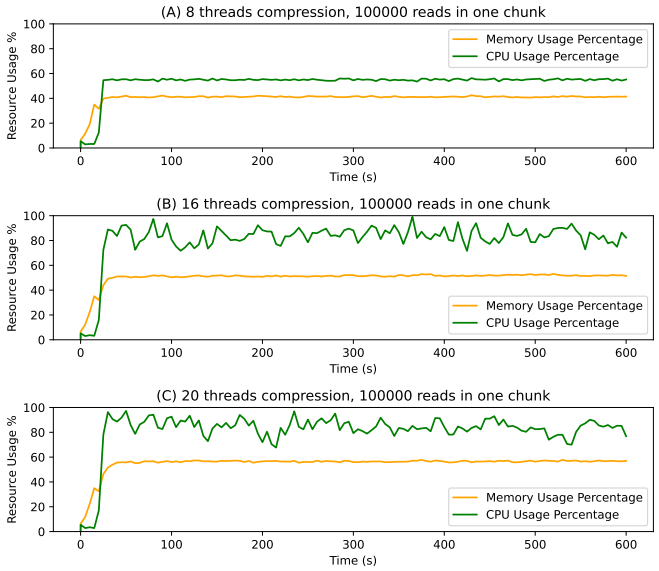

Fig. 19. Memory and CPU usage in compression. Tests are run on a **ecs.c7se.4xlarge** cloud server with 16 CPUs and 32 GB memory; the dataset is E100024251_L01_104.

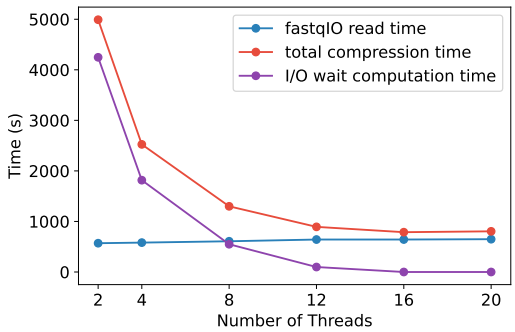

Fig. 20. Scalability evaluation of our algorithm: The evaluation is performed on **ecs.g7.32xlarge** with sufficient CPUs and memory.

Lastly, we analyze the scalability of our algorithm by recording compression wall time as we increase the number of threads. As shown in Fig. 20, our algorithm scales well when there are fewer than 16 threads. The I/O wait computation time decreases to 0 when there are 20 threads, meaning the I/O has been too slow to provide sufficient data for so many threads to consume. If the I/O is faster than the computation, the read buffer will build up to full and the I/O has to wait for the computation to finish to continue reading data. The total compression time converges with fastqIO read time when enough threads are provided. As the genome sequence data is currently distributed by gzip or plain text format, there is very little room to optimize for parallel I/O. However, we will show that our method to compress floating-point tensors can scale up further with faster parallel I/O on supercomputers in the next subsection.

5.3 Floating-point Tensor Compression Evaluation

We first evaluate the relationship between layer depth and compression ratio for our proposed layer-by-layer compression algorithm on the four 3D floating-point tensor applications. As Fig. 21 shows, the compression ratio increases when the layer gets thicker for most datasets. This is because

the 3D tensor gives the compressor more information to predict nearby data values. The current SZ3 3D interpolation method can reach a higher compression ratio for 3D data. However, we also notice that after the thickness reaches 32 for Miranda and Nyx, the compression ratio does not increase clearly. That is, when applying such a thin layer, we can already reach a good compression ratio with a small amount of memory or a high level of parallelization. The Forced Isotropic Turbulence result shows that for certain huge datasets, the 2D layer already contains many data points for prediction, which might even outperform 3D compression in terms of compression ratio.

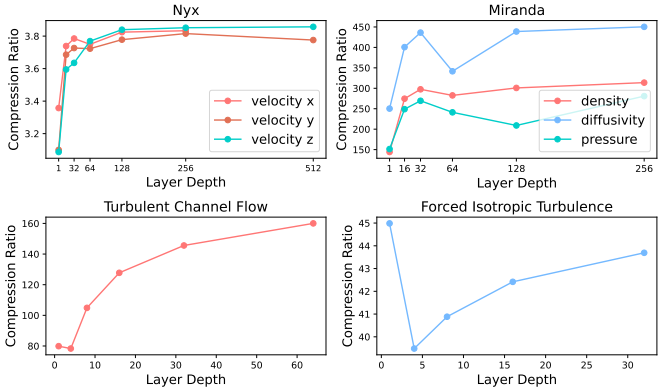

Fig. 21. Compression ratio vs. Layer Depth: Multiple fields in Nyx, Miranda and two large tensors Turbulent Channel Flow and Forced Isotropic Turbulence.

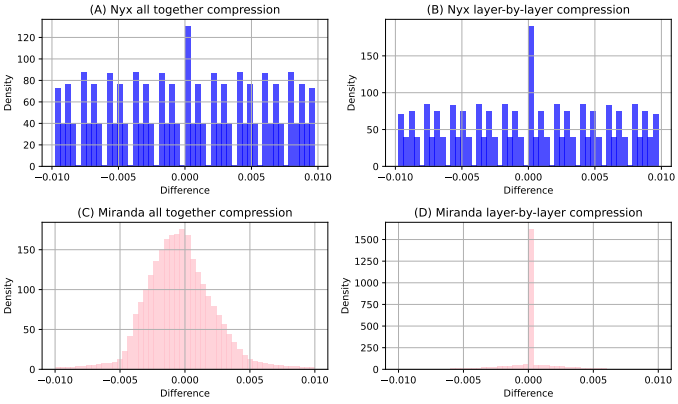

Fig. 22. Nyx temperature and Miranda density pointwise error distribution. The error bound is set to 0.01 for all four configurations. The layer depth is set to 32 for layer-by-layer compression. The compression ratios are (A) 2.17 (B) 2.07 (C) 313, (D) 297.

We then evaluate the compressed data quality with point-wise error distribution and visualization. The experiment result shows that our layer-by-layer compression method has comparable or even superior compression quality compared to the traditional all-together method. The pointwise error distribution changes when using layer-by-layer compression: see Fig. 22. This approach appears to have a more concentrated error for both Miranda and Nyx datasets, although it has a slightly worse compression ratio due to the overhead of describing each layer. We can also verify the compression quality via the visualization method provided by Ocelot in Fig. 23, where there is no obvious

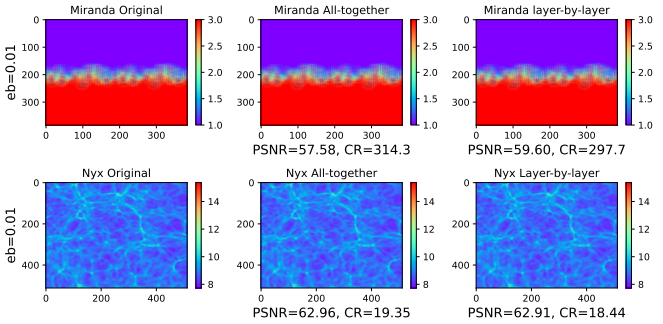

Fig. 23. Visualization of (a) Miranda density and (b) Nyx temperature (log scale). We arbitrarily choose layer 128 for Miranda and layer 488 for Nyx to show the visualization results, as any layers look quite similar to human eyes. The layer depth in layer-by-layer compression is 32. Axes correspond to the X and Y spatial dimensions (384x384 for Miranda and 512x512 for Nyx).

difference between the original data and our compressed data. The PSNR values are also quite close in the two methods.

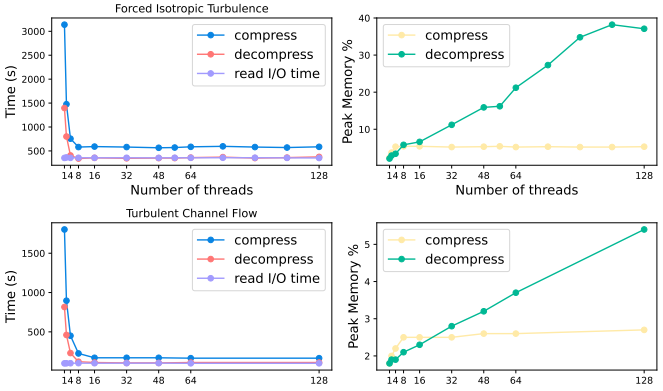

Fig. 24. Compression time vs number of threads for the two large datasets. The layer depth is set to 4.

Next, we evaluate the scalability of our multi-threaded layer-by-layer compression algorithm. Fig. 24 shows that our algorithm can benefit from multiple CPU cores to compress a huge file in a much shorter time. The total compression wall time ceases to decrease after reaching 16 threads because the I/O has become the bottleneck. The program spends 160 seconds compressing 270GB of data, while the total amount of read time is 100s, the overhead of multi-thread is quite minimal. To further improve the performance, parallel I/O is needed as the read speed has become a bottleneck. Also, we note that the memory consumption is higher in decompression with more threads while the memory consumption is higher in compression with fewer than 16 threads. This is because the write I/O becomes a bottleneck in decompression and many worker threads hold the decompressed data to be written. This can cause an out-of-memory error with a mismatched I/O speed and number of threads.

We further improve the scalability by utilizing MPI programming with parallel I/O support and multi-node coordination. To avoid the out-of-memory problem for Forced Isotropic Turbulence and optimize (de)compression time, we limit the tasks-per-node parameter to 32 but increase the number of nodes to get more CPU cores, e.g. we used

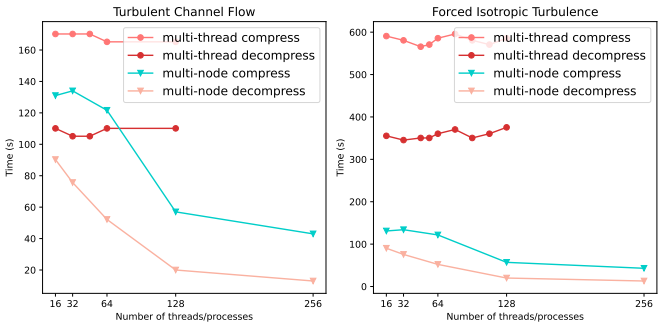

Fig. 25. Compression performance comparison between single-node multi-threaded and multi-node multi-process methods. The multi-thread line stops at 128 threads because the memory is insufficient with more threads on a single node and the program would exit with an out-of-memory error.

8 nodes to get 256 processors. Also, we keep the ratio of the number of I/O processes and the number of worker processors 1:8 to reach near-optimal performance. As Fig. 25 shows, the (de)compression time continues to decrease after using more CPU cores. We also notice that this approach's total time is much lower than the multi-threading method even with the same number of threads/processes. It is largely because we use a more proper ratio of I/O processes and worker processes. In the multi-threading model, when the worker threads are more than the read thread's ability to fill up the read queue, workers may contend for locks with a higher overhead time. Most existing error-bounded compressors have no such capability to parallelize to this scale. Our approach extends floating-point tensor compression to non-uniform memory access systems.

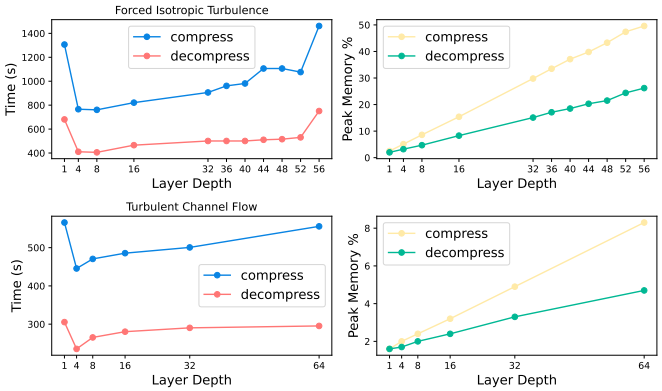

Fig. 26. Compression time and memory consumption of 4-thread layer-by-layer compression on Turbulent Channel Flow dataset. Experiment on Purdue Anvil **shared** partition with 4 tasks per node and 512 GB memory.

Another advantage of our method is that it demands minimal memory for compression, particularly with thin layers. As shown in Fig. 26, the layer depth determines the peak memory consumption. For a thin layer with depth 1, the program only needs 1%, 5 GB memory to compress a file of over 900GB. Other compressors like SZ3, SZ, ZFP, and MGARD face challenges with large tensors, primarily due to memory limitations. On the other hand, our program can run on multiple nodes to utilize memory on multiple nodes with thicker layers to obtain a better compression ratio.

5.4 Compression Ratio/Time Estimation

To make an estimation of compression time and ratio, we apply a decision tree regressor model on 11 features described in Section 4.3. We set various different error bounds from 1e-6 to 1e-1 to compress the data and collect the features. We then train a decision tree regressor using 70% of the data for training and 30% for testing on each of the applications in TABLE 4.

The distribution of the difference between the predicted values and real values is shown in Fig. 27. The green bounding box shows the 80% confidence interval, meaning 80% of prediction error falls into the green box. Thinner box means higher prediction accuracy. Fig. 27 indicates our prediction method performs very well, as the differences between predicted and actual values are very close to 0.

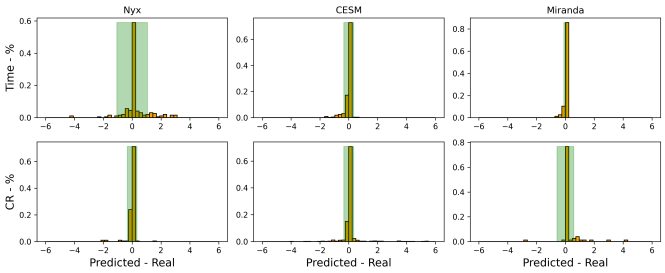

Fig. 27. Nyx/CESM/Miranda application compression time and ratio prediction error distribution (measured on Bebop KNL partition): the X-axis is the difference between the predicted value and the real value, the Y-axis is the percentage for each small range of difference values.

The prediction has a negligible overhead (around 1.7%) compared with the total compression time when we sample 1% of data (using 1 data point every 100 data points). As shown in Fig. 28 (A), the sampling helps reduce the overhead time from more than 70% to less than 5%. The extracted compressor-based features p_0 and P_0 are different from the actual percentage of the zero quantization code because we run the Lorenzo prediction with the real data values instead of the reconstructed data values.

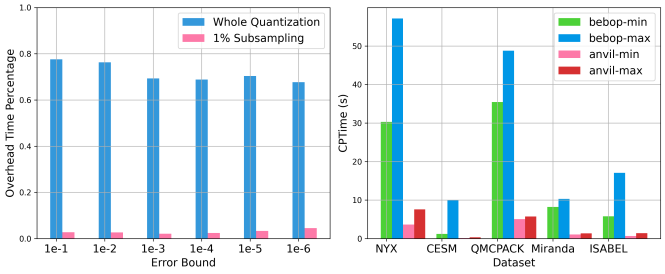

Fig. 28. (A) Overhead time analysis on Nyx application; (B) Compression time range on Bebop and Anvil machines for multiple applications.

TABLE 7 shows the prediction results for our datasets. We can observe from the values that the compression time is gathered into groups related to the application to which they belong. Moreover, we see that our model can always precisely predict the compression ratio and time at different error-bound settings. This is because the distribution of the quantization code changes according to error bounds, and our model captures this information with p_0, P_0 and the quantization entropy effectively.

TABLE 6
Data compression and transfer performance. T(NP): transfer time with no compression, Speed(NP): transfer speed with no compression, CPTime: time taken to compress the data, T(CP): transfer time for compressed data, Speed(CP): transfer speed for compressed data, DPTime: time taken to decompress data, Total: sum of compression time, transfer time, and decompression time; Reduced: total time reduced with compression. All times in seconds.

Dataset	Total Size	Direction	T(NP)	Speed(NP)	CPTime	T(CP)	Speed(CP)	DPTime	Total	Reduced
Nyx	3.22 GB	Anvil→Rockfish	6	547 MB/s	20	38	12.3 MB/s	20	78	-72
		Anvil→ECS	273	11.8 MB/s	20	42	9.49 MB/s	40	102	171
		ECS→Anvil	273	10.3 MB/s	60	45	10.3 MB/s	20	125	148
Turbulent Channel Flow	256 GB	Anvil→Rockfish	774	355 MB/s	175	55	338 MB/s	735	965	-191
		Anvil→ECS	25303	10.4 MB/s	175	1740	11.0 MB/s	134	2049	23254
		ECS→Anvil	23198	11.3 MB/s	230	1709	11.2 MB/s	125	2064	21134
Genome A	39.27 GB	Rockfish→Anvil	64	611 MB/s	1095	36	316 MB/s	375	1506	-1442
		Anvil→(Rockfish)→ECS	3527	11.4 MB/s	**N/A**	1055	11.3 MB/s	874	**3088**	439
		ECS→(Rockfish)→Anvil	3904	10.3 MB/s	1100	1181	10.1 MB/s	**N/A**	**2692**	1212

TABLE 7
Compression Time and Ratio Prediction Examples: EB denotes error bound, CR denotes compression ration, CPTime denotes compression time. P-CR and P-CPTime denote predicted compression ratio and compression time, respectively. All time-related information is measured in seconds and collected on the Bebop KNL partition.

Dataset	**EB**	**P-CR**	**CR**	**P-CPTime**	**CPTime**
Nyx Baryon Density	1e-6	1.19	1.18	35.90	35.60
	1e-4	3.15	3.10	32.30	33.30
	1e-2	10.40	10.20	30.30	30.30
CESM LHFLX	1e-6	1.14	1.14	1.46	1.46
	1e-3	2.56	2.49	1.97	1.59
	1e-2	5.25	4.43	1.55	1.50
CESM SNOWHICE	1e-6	5.36	6.97	1.61	1.85
	1e-4	21.00	21.90	1.55	1.58
	1e-3	48.00	52.80	1.40	1.48
RTM-1982	1e-6	4.78	4.80	13.85	13.32
RTM-1048	1e-4	24.72	24.89	13.10	13.30
RTM-0594	1e-4	83.15	84.99	12.13	11.43
Miranda Velocity-x	1e-2	18.99	16.74	9.57	9.31
	1e-3	7.11	7.67	10.17	9.70
	1e-1	9.11	9.43	52.05	52.49

5.5 Data Transfer Performance

Data transfer performance is crucial in determining whether data compression is advantageous for achieving a shorter transfer time, considering the additional time costs incurred during compression and decompression. Our framework aims to be adaptable to various network conditions to meet users' requirements. Generally, supercomputers boast extensive network bandwidths, whereas cloud service providers like Alibaba Cloud may provide a more restricted network capability. We evaluate the data transfer performance of our Ocelot framework on the machines listed in Table 2.

The findings suggest that our proposed compression algorithms offer significant advantages for systems with slow networks and rapid computation. As demonstrated in Table 6, the performance improvement exceeds 10x when transferring data from supercomputers to typical cloud computing clusters. While contemporary cloud computing platforms boast comparable single-node computational capabilities to supercomputers, many are constrained by network bandwidth limitations. Our framework holds the potential to substantially enhance transfer efficiency across these platforms.

On the other hand, the overall transfer time may not benefit from compression in supercomputers because of superior networks. Nonetheless, our Ocelot framework effectively leverages specific characteristics to distribute computational tasks across different supercomputers. For example, we aim to transmit genome sequence data from Purdue Anvil to the Alibaba ECS. Due to the absence of Intel AVX512 optimization in the AMD Zen 3 processors utilized by Purdue Anvil, our genome sequence compression code cannot be compiled on this cluster. As a workaround, we transfer the data to Rockfish for compression before forwarding the compressed data to the ECS destination. This roundtrip approach ultimately optimizes overall time efficiency despite the intermediate steps involved, as shown in the third row of Table 6.

6 Conclusion and Future Work

We developed a novel interactive, real-time Compression-as-a-Service (CaaS) platform Ocelot that focuses on data compression with optimized strategies among multiple computing clusters. Based on our experiments on compressing, transferring, and storing the floating-point tensors and genome sequence datasets, we report the following key findings.

- Visualization and data preview provide a intuitive way for users to set multiple error bounds on interesting regions/ranges. Ocelot provides a user-friendly graphical user interface lacking in most previous compressors.
- Ocelot can significantly benefit overall transfer performance for computing clusters with good computing power but slower networks. It can automatically offload computing tasks to other clusters with its remote orchestration.
- Our proposed layer-by-layer compression uses a series of 2D or relatively thin 3D layers with prediction-based compression. The method avoids the memory limit constraint and offers great parallel compression capability.
- Our genome sequence compression algorithm achieves better compression ratios than state-of-the-art algorithms, by employing a better alignment algorithm and an optimized quality score compression method, and by not keeping string identifiers and read orders.

Ocelot orchestrates compression on multiple clusters and is not limited to the aforementioned data types. Future reseach can integrate more types of compressors and even

other computing tasks into the framework. Our compression run time prediction is rather simple and relies on historical data. It does not adapt to the characteristics of different compressors. Future work can look into the compressor details to see if more accurate predictions can be made with limited overhead.

ACKNOWLEDGMENTS

The material was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (ASCR), under contract DE-AC02-06CH11357, and supported by the National Science Foundation under Grant OAC-2003709 and OAC-2104023. We acknowledge the computing resources provided on Bebop (operated by Laboratory Computing Resource Center at Argonne) and also ACCESS-CI computing resources (Purdue Anvil and Rockfish).

REFERENCES

[1] T. E. Fornek, "Advanced photon source upgrade project preliminary design report," 9 2017.

[2] N. Tchipev *et al.*, "Twetris: Twenty trillion-atom simulation," *The International Journal of High Performance Computing Applications*, vol. 33, no. 5, pp. 838–854, 2019.

[3] K. Chard, S. Tuecke, and I. Foster, "Globus: Recent enhancements and future plans," in *XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, ser. XSEDE16. New York, NY, USA: Association for Computing Machinery, 2016.

[4] Y. Liu, Z. Liu, R. Kettimuthu, N. Rao, Z. Chen, and I. Foster, "Data transfer between scientific facilities – bottleneck analysis, insights and optimizations," in *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2019, pp. 122–131.

[5] D. Lan, R. Tobler, Y. Souilmi, and B. Llamas, "Genozip: a universal extensible genomic data compressor," *Bioinformatics*, vol. 37, no. 16, pp. 2225–2230, 02 2021. [Online]. Available: https://doi.org/10.1093/bioinformatics/btab102

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

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

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

[9] D. Tao, S. Di, Z. Chen, and F. Cappello, "In-depth exploration of single-snapshot lossy compression techniques for n-body simulations," in *IEEE International Conference on Big Data*, 2017, pp. 486–493.

[10] X.-C. Wu, S. Di, E. M. Dasgupta, F. Cappello, H. Finkel, Y. Alexeev, and F. T. Chong, "Full-state quantum circuit simulation by using data compression," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356155

[11] X.-C. Wu, S. Di, F. Cappello, H. Finkel, Y. Alexeev, and F. T. Chong, "Amplitude-aware lossy compression for quantum circuit simulation," 2018. [Online]. Available: https://arxiv.org/abs/1811.05140

[12] X. Liang, S. Di, D. Tao, S. Li, B. Nicolae, Z. Chen, and F. Cappello, "Improving performance of data dumping with lossy compression for scientific simulation," in *IEEE International Conference on Cluster Computing*, 2019, pp. 1–11.

[13] 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 *IEEE International Conference on Cluster Computing*, 2018, pp. 1–11.

[14] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[15] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[16] K. Zhao, S. Di, M. Dmitriev, T.-L. D. Tonellot, Z. Chen, and F. Cappello, "Optimizing error-bounded lossy compression for scientific data by dynamic spline interpolation," in *IEEE 37th International Conference on Data Engineering*, 2021, pp. 1643–1654.

[17] J. Liu, S. Di, K. Zhao, S. Jin, D. Tao, X. Liang, Z. Chen, and F. Cappello, "Exploring autoencoder-based error-bounded compression for scientific data," in *IEEE International Conference on Cluster Computing*, 2021, pp. 294–306.

[18] X. Liang, B. Whitney, J. Chen, L. Wan, Q. Liu, D. Tao, J. Kress, D. Pugmire, M. Wolf, N. Podhorszki, and S. Klasky, "Mgard+: Optimizing multilevel methods for error-bounded scientific data reduction," 2020.

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

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

[21] M. H. Rahman, S. Di, K. Zhao, R. Underwood, G. Li, and F. Cappello, "A feature-driven fixed-ratio lossy compression framework for real-world scientific datasets," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, 2023, pp. 1461–1474.

[22] A. Ganguli, R. Underwood, J. Bessac, D. Krasowska, J. C. Calhoun, S. Di, and F. Cappello, "A lightweight, effective compressibility estimation method for error-bounded lossy compression," in *2023 IEEE International Conference on Cluster Computing (CLUSTER)*, 2023, pp. 247–258.

[23] R. Underwood, J. Bessac, D. Krasowska, J. C. Calhoun, S. Di, and F. Cappello, "Black-box statistical prediction of lossy compression ratios for scientific data," *The International Journal of High Performance Computing Applications*, vol. 37, no. 3-4, pp. 412–433, 2023. [Online]. Available: https://doi.org/10.1177/10943420231179417

[24] P. J. A. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice, "The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants," *Nucleic Acids Research*, vol. 38, no. 6, pp. 1767–1771, 12 2009. [Online]. Available: https://doi.org/10.1093/nar/gkp1137

[25] Y. Zhang, L. Li, Y. Yang, X. Yang, S. He, and Z. Zhu, "Light-weight reference-based compression of fastq data," *BMC bioinformatics*, vol. 16, pp. 1–8, 2015.

[26] Z.-A. Huang, Z. Wen, Q. Deng, Y. Chu, Y. Sun, and Z. Zhu, "Lw-fqzip 2: a parallelized reference-based compression of fastq files," *BMC bioinformatics*, vol. 18, pp. 1–8, 2017.

[27] Y. Xing, G. Li, Z. Wang, B. Feng, Z. Song, and C. Wu, "Gtz: a fast compression and cloud transmission tool optimized for fastq files," *BMC bioinformatics*, vol. 18, no. 16, pp. 233–242, 2017.

[28] G. Benoit, C. Lemaitre, D. Lavenier, E. Drezen, T. Dayris, R. Uricaru, and G. Rizk, "Reference-free compression of high throughput sequencing data with a probabilistic de bruijn graph," *BMC bioinformatics*, vol. 16, no. 1, pp. 1–14, 2015.

[29] D. C. Jones, W. L. Ruzzo, X. Peng, and M. G. Katze, "Compression of next-generation sequencing reads aided by highly efficient de novo assembly," *Nucleic acids research*, vol. 40, no. 22, pp. e171–e171, 2012.

[30] R. Chard, Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, and K. Chard, "Funcx: A federated function serving fabric for science," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 65–76. [Online]. Available: https://doi.org/10.1145/3369583.3392683

[31] T. Hacker, B. Athey, and B. Noble, "The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network," in *Proceedings 16th International Parallel and Distributed Processing Symposium*, 2002, pp. 10 pp–.

[32] E. Yildirim, J. Kim, and T. Kosar, "How GridFTP pipelining, parallelism and concurrency work: A guide for optimizing large

dataset transfers," in *SC Companion: High Performance Computing, Networking Storage and Analysis*, 2012, pp. 506–515.
[33] A. Bookstein, V. Kulyukin, and T. Raita, "Generalized hamming distance," *Information Retrieval*, vol. 5, 10 2002.
[34] C. Zhao, "String correction using the damerau-levenshtein distance," *BMC Bioinformatics*, 06 2019.
[35] M. Sardaraz, M. Tahir, A. A. Ikram, and H. Bajwa, "Seqcompress: An algorithm for biological sequence compression," *Genomics*, vol. 104, no. 4, pp. 225–228, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888754314001499
[36] U. Ghoshdastider and B. Saha, "Genomecompress: A novel algorithm for dna compression," 2007.
[37] S. Marco-Sola, J. M. Eizenga, A. Guarracino, B. Paten, E. Garrison, and M. Moreto, "Optimal gap-affine alignment in O(s) space," *Bioinformatics*, vol. 39, no. 2, p. btad074, 02 2023. [Online]. Available: https://doi.org/10.1093/bioinformatics/btad074
[38] S. Marco-Sola, J. C. Moure, M. Moreto, and A. Espinosa, "Fast gap-affine pairwise alignment using the wavefront algorithm," *Bioinformatics*, vol. 37, no. 4, pp. 456–463, 09 2020. [Online]. Available: https://doi.org/10.1093/bioinformatics/btaa777
[39] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, and . G. P. D. P. Subgroup, "The Sequence Alignment/Map format and SAMtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 06 2009. [Online]. Available: https://doi.org/10.1093/bioinformatics/btp352
[40] Facebook, "Zstandard," https://github.com/facebook/zstd/releases.
[41] M. V. Mahoney, "The zpaq compression algorithm," 2015, https://api.semanticscholar.org/CorpusID:13248511.
[42] D. Feng, T. Li, G. H. Li, and X. Wang, "Reverse time migration of gpr data based on accurate velocity estimation and artifacts removal using total variation de-noising," *Journal of Applied Geophysics*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:246657925
[43] NYX simulation, https://amrex-astro.github.io/Nyx, online.
[44] Miranda, https://wci.llnl.gov/simulation/computer-codes/miranda.
[45] X. Hongxin, "DNBSEQT7 WES-PE150 demo data," https://db.cngb.org/search/project/CNP0003660/, 10 2022.
[46] "BGISEQ500 PCRfree NA12878 CL100076243 L01," https://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/data/NA12878/BGISEQ500/.
[47] "DNBSEQ-T7 WES PE150 ," https://db.cngb.org/search/experiment/CNX0547764/.
[48] QMCPack, https://qmcpack.org/, online.
[49] Hurricane ISABELA Simulation Datasets, http://vis.computer.org/vis2004contest/data.html.
[50] J. Kay, C. Deser, A. Phillips, A. Mai, C. Hannay, G. Strand, J. Arblaster, S. Bates, G. Danabasoglu, J. Edwards *et al.*, "The community earth system model (CESM), large ensemble project: A community resource for studying climate change in the presence of internal climate variability," *Bulletin of the American Meteorological Society*, vol. 96, no. 8, pp. 1333–1349, 2015.
[51] M. Lee and R. D. Moser, "Direct numerical simulation of turbulent channel flow up to $Re_\tau \approx 5200$," *Journal of Fluid Mechanics*, vol. 774, pp. 395–415, jul 2015.
[52] S. Chandak, K. Tatwawadi, I. Ochoa, M. Hernaez, and T. Weissman, "Spring: a next-generation compressor for fastq data," *Bioinformatics*, Aug 2019.

Yuanjian Liu is a Ph.D. student at the University of Chicago. His research interests include autonomous laboratories, computer vision, high-performance computing, and the utilization of scientific data. He is also working on the availability of knowledge and believes that online education can have both higher efficiency and broader influence. Email: yuanjian@uchicago.edu.

Sheng Di (Senior Member, IEEE) received his master's degree from Huazhong University of Science and Technology in 2007 and Ph.D. degree from the University of Hong Kong in 2011. He is currently a computer scientist at Argonne National Laboratory. His research interests involve resilience on high-performance computing (such as silent data corruption, optimization checkpoint model, and in situ data compression) and broad research topics on cloud computing. He is working on multiple HPC projects, such as detection of silent data corruption, characterization of failures and faults for HPC systems, and optimization of multilevel checkpoint models. He is the recipient of a DOE 2021 Early Career Research Program award. Email: sdi@anl.gov.

Jiajun Huang is a Ph.D. candidate in Computer Science at the University of California, Riverside, and a long-term visiting student at Argonne National Laboratory. He received his bachelor's degree in Electronic Information Engineering from the University of Electronic Science and Technology of China (UESTC) and the University of Glasgow (Honors of the First Class), in 2021. He has published over 10 papers in top-tier conferences and journals, including SC, ICS, IPDPS, and TPDS. He has also received several prestigious honors, including the First Place Award in the ACM Student Research Competition (Graduate) at SC '23 and the Dissertation Completion Fellowship Award (DCFA) from UC Riverside. His research interests include high-performance computing and communication, high-performance deep learning, parallel & distributed computing, and big data management & analytics. Email: jhuan380@ucr.edu

Zhaorui Zhang is currently a research assistant professor in the Department of Computing at The Hong Kong Polytechnic University. She received her Ph.D. from the Department of Computer Science at The University of Hong Kong, Hong Kong, and her BSc degree in computer science from Xi'an Jiaotong University. Her research interests include distributed machine learning systems, distributed systems, HPC, cloud computing, and data reduction. Email: zhaorui.zhang@polyu.edu.hk.

Kyle Chard is a research assistant professor in the Department of Computer Science at the University of Chicago. He also holds a joint appointment at Argonne National Laboratory. He received his Ph.D. in computer science from Victoria University of Wellington, New Zealand, in 2011. He is a member of the ACM and IEEE. He co-leads the Globus Labs research group, which focuses on a broad range of research problems in data-intensive computing and research data management. Email: chard@uchicago.edu.

Ian Foster is an Argonne Distinguished Fellow, senior scientist, and director of the Data Science and Learning division at Argonne National Laboratory and a professor in the Department of Computer Science at the University of Chicago. He develops tools and techniques that allow people to use high-performance computing technologies to do qualitatively new things. His work involves investigations of parallel and distributed languages, algorithms, and communication, as well as applications. He is particularly interested in using high-performance networking to incorporate remote compute and information resources into local computational environments. Email: foster@cs.uchicago.edu.