DOI: 10.14529/jsfi140205

Data Compression for the Exascale Computing Era – Survey

Seung Woo Son[1], *Zhengzhang Chen*[1], *William Hendrix*[1], *Ankit Agrawal*[1], *Wei-keng Liao*[1], *and Alok Choudhary*[1]

While periodic checkpointing has been an important mechanism for tolerating faults in high-performance computing (HPC) systems, it is cost-prohibitive as the HPC system approaches exascale. Applying compression techniques is one common way to mitigate such burdens by reducing the data size, but they are often found to be less effective for scientific datasets. Traditional lossless compression techniques that look for repeated patterns are ineffective for scientific data in which high-precision data is used and hence common patterns are rare to find. In this paper, we present a comparison of several lossless and lossy data compression algorithms and discuss their methodology under the exascale environment. As data volume increases, we discover an increasing trend of new domain-driven algorithms that exploit the inherent characteristics exhibited in many scientific dataset, such as relatively small changes in data values from one simulation iteration to the next or among neighboring data. In particular, significant data reduction has been observed in lossy compression. This paper also discusses how the errors introduced by lossy compressions are controlled and the tradeoffs with the compression ratio.

Keywords: Fault tolerance, checkpoint/restart, lossless/lossy compression, error bound, data clustering.

1. Introduction

The future extreme scale computing systems [13, 35] are facing several challenges in architecture, energy constraints, memory scaling, limited I/O, and scalability of software stacks. As the scientific simulations running on such systems continue to scale, the possibility of hardware/software component failures will become too significant to ignore. Therefore, a greater need is emerging for effective resiliency mechanisms that consider the constrains of (relatively) limited storage capability and energy cost of data movement (within deep memory hierarchy and off-node data transfer). The traditional approach to storing checkpoint data in raw formats will soon be cost-prohibitive. On the other hand, storing the states of simulations for restart purposes will remain necessary. Thus, in order to scale resiliency via the checkpoint/restart mechanism, multiple dimensions of the problem need to be considered to satisfy the constraints posed by such an extreme-scale system.

A popular solution to reduce checkpoint/restart costs is to apply data compression. However, because scientific datasets are mostly floating-point numbers (in single or double precision), a naive use of compression algorithms can merely bring a limited improvement in terms of the amount of data reduced while bearing a high compression overhead to perform compression. To tackle such problem of hard-to-compress scientific data, recently-proposed compression algorithms start to explore data characteristics exhibited in specific domains and develop ad-hoc strategies that perform well on those data. The motivation comes from the fact that most of the simulations calculates values on points (nodes, particles, etc.) in a discretized space to solve its own mathematical models and continues along the temporal dimension until a certain stop condition is met. One can potentially exploit the patterns exhibited along both spatial and temporal dimensions in improving effectiveness of existing compression algorithms because the variance of data in the neighborhood tends to be small in many cases. Existing compression algorithms seldom consider such data patterns.

[1]Department of EECS, Northwestern University, Evanston, USA

Lossy compressions [11, 15] often produce better compression ratios than the lossless counterpart, but the error rates are not easy to bound, and, in large scale simulations, unbounded error could introduce significant deviation from the actual values, leading to impact the outcome of the simulation. In other words, lossy compressions could make compressed data useless. Lossy compression on checkpointing implies several challenges for large-scale simulations, e.g., guaranteeing point-wise error bounds defined by the user, reducing a sufficiently large amount of storage space, performing compression in-situ (to reduce data movement), and taking advantages of data reduction by potentially being able to use locally-available non-volatile storage devices. In this paper, we survey a set of compression techniques and discuss whether they can achieve the above goals. We also describe ideas of making use of machine learning techniques to discover temporal change patterns, designing a data representation mechanism to capture the relative changes, and solutions to meet the user request on tolerable error bound.

We anticipate that checkpointing will continue to be a crucial component of resiliency and lossy compressions will become an acceptable method to reduce the data size at the runtime. The critical point is whether the data can be significantly reduced given a controlled error bound. Such an error tolerance will be specified by the user as an input parameter. For example, a user can indicate that maximum tolerable error per point of 0.1%, and lossy compression algorithms must guarantee that bound. Overall, applying data compression at runtime for checkpointing will become appealing to large-scale scientific simulations on future high-performance computing systems, as it can reduce storage space as well as save the energy resulted from data movement.

The remainder of this paper is organized as follows. The discussion of lossless compression algorithms for traditional checkpointing mechanisms is described in Section 2. Section 3 presents several studies to apply lossy compression algorithms on scientific applications. Finally, we conclude the paper in Section 4.

2. Lossless Compression

Compression has two advantages. At first, it can reduce the space required to store the data. Secondly, it can improve I/O bandwidth (disk or network communication), as the time spent on data checkpointing is reduced. However, compression comes with the CPU and memory overhead to compress and decompress the data. Thus, the effectiveness of applying data compression is determined by the achievable compression ratio of the selected compression algorithm and the time to compress the data. The future exascale systems are expected to exhibit a trend that the data movement among memory hierarchies and off-node data transfer will become relatively expensive in terms of time and energy, compared to the ever-increasing compute power. If the selected compression algorithm can produce a reasonable compression ratio, the benefit of applying data compression shall become more significant for the scientific applications running on those systems.

When it comes to compression, scientists often face the dilemma of choosing lossless compression and lossy compression. The former preserves data fidelity but can be slow and produce a poor compression ratio, whereas the latter introduces a cumbersome validation process on the approximated data. In this section, we first compare several existing lossless compression methods and their effectiveness when applied on the scientific datasets.

2.1. Integration within Checkpointing Framework

Welton *et al.* [36] have integrated and evaluated several lossless compression algorithms within the context of IOFSL, an I/O middleware. They evaluate their framework on using commonly-used compression algorithms on synthetic and real datasets. The experimental results show about 1.6x of compression ratio for the air/sea flex data from NCAR data archive. The framework could potentially serve as a baseline I/O utility for HPC systems; however, the algorithms they selected performed rather poorly on scientific data. Ibtesham *et al.* [17] also did a viability study that compares several lossless compression algorithms with the Berkeley Lab Checkpoint/Restart (BLCR) [14] framework. They essentially observed up to 2x compression ratio for several scientific datasets.

Islam *et al.* [18] present mcrEngine, a software framework that first merges compatible data in terms of data type (double, float, other) between checkpoint files and compresses the output of the merged data with an existing compression algorithm. The authors tested mcrEngine on five different codes that exhibited this type of compatibility between checkpoint files and compressed the checkpoint data by a factor of about 1.2. While this checkpoint-level similarity may not be applicable to every simulation code, the technique of combining this data before compression clearly improves compressibility.

Several high-level I/O libraries provide a compression capability in a transparent manner. HDF5 requires users to use chunking to create a compressed dataset. Currently HDF5 provides two predefined compression filters (zlib and szip). All other compression filters must be registered with HDF5 before they can be used [34]. ADIOS [27] also provides a similar compression capability as a data transformation mechanism in file-based I/O. ADIOS provides three common compression algorithms (zlib, gzip, and szip) as well as ISOBAR [30], which could potentially yield higher compression ratio on scientific datasets.

There are recent studies about exploiting compute power in SSDs to perform in-situ operations including compression. Active Flash by Boboila *et al.* [7] is such an example where the compression is performed within SSDs in the context of data analytic pipelines. Since their main objective is to determine feasibility of performing in-situ on SSDs, they also simply use existing lossless compression algorithm, LZO, on common HPC datasets: atmospheric and geosciences (NetCDF format) and bioinformatics (text format). Active Flash achieved about 1.96x of compression ratio for both datasets.

2.2. Increasing Compressibility through Transformations

Bicer *et al.* [6] propose a novel compression algorithm for climate data, CC (climate compression), that takes advantage of the spatial and temporal locality inherent in climate data to accelerate the storage and retrieval of data. Their methodology uses an exclusive or (`xor`) of adjacent or consecutive data to reduce the entropy in the data, which is analogous to taking a difference or ratio in the sense that similar data values will neutralize to form easily compressed datasets.

Schendel *et al.* propose the ISOBAR [29, 30] framework where it first divides the data into compressible and incompressible segments before applying lossless compression to reduce data size. PRIMACY [31] and ALACRITY [19] applied similar data transformation methods on byte columns in order to identify compressible byte segments. We note that ISOBAR used FPC [8], a fast and effective lossless compression algorithm designed for double-precision floating point

Table 1. Comparison of lossless compression schemes.

Scheme	Transformation Applied	Algorithm	Compression Ratio
FPC [8]	not used	it first predicts values sequentially using two predictors (FCM and DFCM), and subsequently selects the closer predicted value to the actual. Lastly, it XORs the selected predicted value with the actual value, and leading-zero compresses the result.	1.02x∼1.96x
ISOBAR [30]	divide byte-columns into compressible and incompressibles	apply zlib, bzlib2, (fpzip, FPC) on all compressible (after discarding noisy byte-columns). zlib is the main compression algorithm; others are for comparison purposes	1.12x∼1.48x
PRIMACY [31]	frequency based permutation of ID values	apply zlib on transformed data	1.13x∼2.16x
ALACRITY [19]	split floating-point values into sign, exponent, and significands	unique-value encoding of the most significant bytes (assuming high-order bytes (sign and exponents) are easy to compress); low-order bytes are compressed using ISOBAR	1.19x∼1.58x
CC [6]	XOR on Δ of neighboring data point in the same iteration	apply zero-filled run length encoding	up to 2.13x
IOFSL [36]	not used	integration of LZO, bzip2, zlib within the I/O forwarding layer	∼1.9x
Binary Masking [5]	bit masking (XOR)	apply zlib on bit masked data in order to partially decreases the entropy level	1.11x∼1.33x
MCRENGINE [18]	variable merging in the same group	apply parallel gzip on the merged variables across processes	up to 1.18x

numbers. FPC first predicts values sequentially using predictors, and subsequently selects the closer predicted value to the actual. It then XORs the selected predicted value with the actual value such that more leading zeroes in the predicted values, which helps improve compression ratios.

Bautista-Gomez and Capello [5] propose an algorithm related to ISOBAR in that both are lossless compression algorithms that seek to identify low-entropy segments of floating-point data and compress them separately. Bautista-Gomez and Capello, however, transform the data by applying a bit masking (`xor`) to the original data in order to reduce its entropy before compression. They present results for a number of scientific datasets (GTC dataset in single-precision and climate dataset in double-precision), achieving a maximum of around 40% data reduction.

2.3. Comparison

Table 1 summarizes the comparison of lossless compression schemes surveyed in this paper. Since lossless algorithms do not incur any loss of information in the uncompressed data, we mainly compare the compression ratio achieved by each method. The compression ratio R for data D of size $|D|$ reduced to size $|D^{'}|$ is denoted as: $R = \frac{|D|}{|D'|}$. We note that scientific simulations use predominantly double-precision floating-point variables. Therefore, the compression ratio presented in Table 1 is for those only, though the algorithm can be applied to floating point numbers of different precision or other types of data. The main takeaway from Table 1 is that the data reduction by lossless compression is limited; the maximum achievable compression ratio is just above 2x, which is also possible after additional data transformation is applied. As

described in the second column of Table 1, most compression algorithms apply some sort of data transformations in order to increase the effectiveness of data compression.

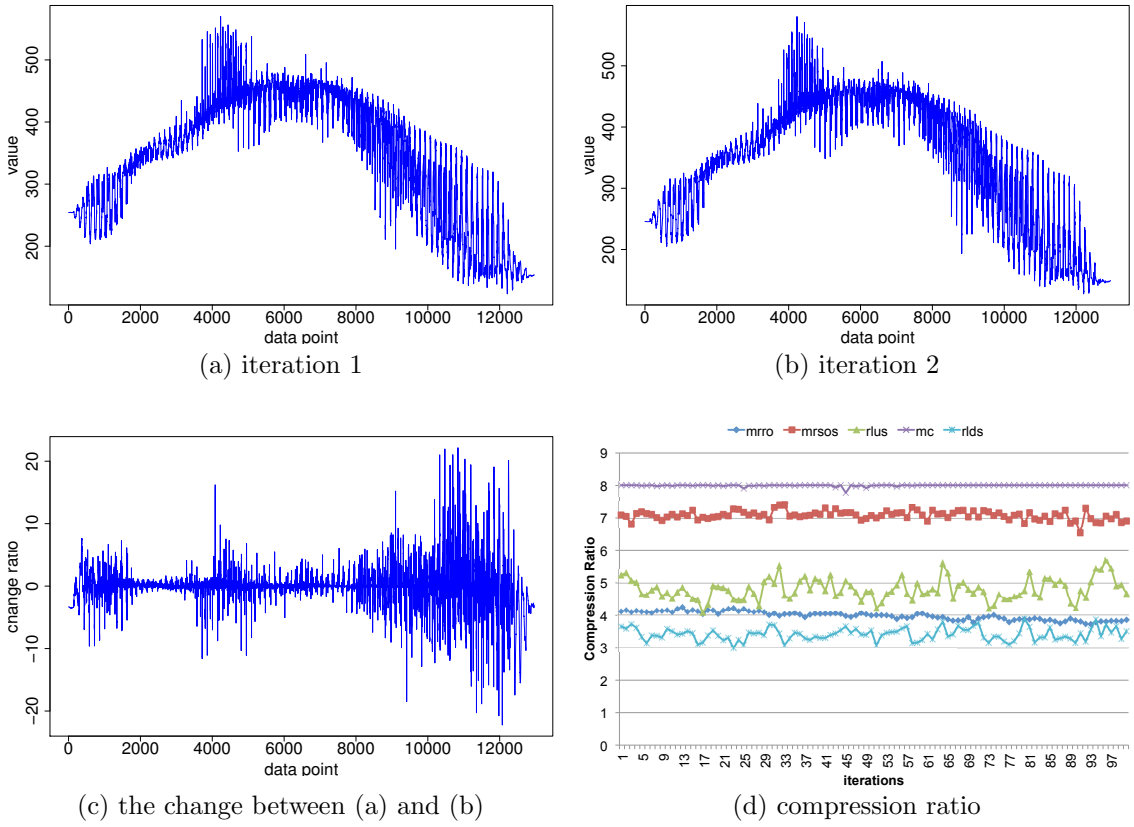
3. Lossy Compression

The intuition behind using lossy compression algorithms stems from the following three observations. First, scientific simulation data, like climate CMIP5 *rlus* data, are considered as one type of high entropy data [12, 28]. Such data often exhibits randomness without any distinct repetitive patterns in one single timestamp or iteration (see Figure 1 (a) or (b)). Thus, traditional *lossless* compression approaches, as described in Section 2, cannot achieve appreciable data reduction. Second, in many scientific applications, relative changes in data values are often not very significant either spatially (among neighboring data points) or temporally (from one simulation iteration to the next). As an example for this temporal similarity, more than 80% of climate rlus data remains unchanged or only change with a percentage less than 0.1% (see Figure 1 (c)). Third, unlike observational data, many scientific simulation codes can tolerate some error-bounded loss in their simulation data accuracy. Thus, *lossy* compression methods can offer some attractive features for data reduction.

The effectiveness of lossy compression however heavily depends on the domain knowledge to select the right compression algorithms, and it is very difficult to get compression beyond a small factor with desired accuracy, not to speak of guaranteeing that the compression error will be smaller than a certain error rate, preferably specified by a user. Figure 1(c) shows the change in data values between two iterations (checkpoints) instead of individual iteration because representing data in change ratio could minimize coding space during compression. For example, a checkpoint with 100 million data points where there are potentially 100 million changes. Two data points where one changes from 10 to 11 and the other from 100 to 110 have the identical relative changes, which can be represented as the same 10 percent change. Therefore, data points with the same change percentage can be indexed by one number. The idea of considering the data changes along temporal domain transforms the data where individual checkpoint seldom exhibits repeated patterns into a space where common patterns in change percentages are easier to find. To ensure the quality of reduced data, one challenge of this approach is to select a set of change values that can represent a large number of neighbors within a small radius, a tolerable error rate specified by the user. This will potentially address the challenge of lossy compression in maintaining the quality of compressed data, which will be discussed in later sections. Furthermore, simulation parameters are often calibrated using data that are themselves subject to measurement error or other inaccuracies. Therefore, very small deviations ($< 1\%$) in restart fidelity are unlikely to hurt normal scientific simulations as long as such deviations are bounded.

3.1. Transformation Schemes

There are handful of prior works studied about applying lossy compression on scientific datasets. Lakshminarasimhan *et al.* [22] described ISABELA, a lossy compression technique based on applying B-splines to sorted data. By transforming the data, ISABELA was able to achieve high compression on data that was previously regarded as "incompressible," while losing minimal data fidelity ($\geq$0.99 correlation with original data). Lakshminarasimhan *et al.* did not



(a) iteration 1 (b) iteration 2

(c) the change between (a) and (b) (d) compression ratio

Figure 1. Data representation of *rlus*, a variable from the CMIP5 climate simulation dataset: (a) original data distribution at iteration 1. (b) original data distribution at iteration 2. (c) the changing percentage of data values between two iterations, and (d) compression ratio ($\frac{\text{original size}}{\text{compressed size}}$) for the clustering-based algorithm on the CMIP5 simulation datasets [33]. Out of the dozens variables available in CMIP5 [1], we randomly chose five, namely *mrsos*, *mrro*, *mc*, *rlds*, and *rlus*. The resolution for these data is 2.5° by 2°. *mc* is a monthly simulation data, while other four are daily data. We note that all approximated data point is within the user-specified error bound, which is 0.1%. The approximation precision used is 8 bits (or 1 byte). The parallel K-means clustering algorithm [3, 20, 25] is used on the temporal change ratios Δ to get $2^B - 1$ clusters, where B is the number of bits.

consider applying ISABELA to checkpoint data because they assume checkpoint data do not permit approximation.

As an another transformation approach, we have also studied a data transformation idea similar to video compression algorithms especially MPEG [24], which stores the differences between successive frames using temporal compression. Similar to MPEG's forward predictive coding where current frames are with reference to a past frame, we code the current checkpoint data based on the previous checkpoint. In order to this, the relative change (or change ratio) is calculated as $\Delta = \frac{D_c - D_p}{D_p}$, where D_c and D_p is the data point in the current and previous iteration, respectively. A potential problem with this approach is that D_p cannot be zero. If D_p is zero, D_c cannot be compressible. This transformation technique is applied to our clustering-based algorithm.

3.2. Approximation Algorithms

Several recent studies [4, 16, 23] have evaluated Samplify's APplication AXceleration (APAX) Encoder along with other lossy compression algorithms on scientific datasets, mostly climate dataset. The APAX algorithm encodes sequential blocks of input data elements with user-selected block size between 64 and 16,384. The signal monitor tracks the input dataset's center frequency. The attenuator multiplies each input sample by a floating-point value that, in fixed-rate mode, varies from block to block under the control of an adaptive control loop that converges to the user's target compression ratio. The redundancy remover generates derivatives of the attenuated data series and determines which of the derivatives encodes using the fewest bits. The bit packer encodes groups of 4 successive samples using a joint exponent encoder (JEE). JEE exploits the fact that block floating-point exponents are highly correlated, which is commonly observed in many lossless compress techniques [8, 19, 29, 30, 31].

fpzip quantizes the significant bits while leaving the values in their floating-point format. *fpzip* fixes the quantization level to be a power of two, thereby effectively truncating (zeroing) a certain number of significant bits. Designed such, *fpzip* can be lossless if the quantization level is the same as the original data representation. ISABELA, on the other hand, applies the B-splines curve fitting (interpolation) to the sorted data. A B-splines curve is a sequence of piecewise lower order parametric curves joined together via knots. Specifically, they used the cubic B-splines for faster interpolation time and producing smooth curves [22].

The clustering-based approach, on the other hand, uses machine learning techniques once the change ratios for all data points are calculated. Once the change ratios of all data points have been calculated, using machine learning techniques, we first calculate the distributions of changes and then approximate them into an indexed space to achieve the goals of maximal data reduction. In our preliminary implementation, we use histogram for learning distribution of the change ratios obtained. Histogram is a popular method to learn the data distribution. For example, data shown in Figure 1(c) can be easily converted into a histogram. Histogram estimates the probability distribution by partitioning the data into discrete intervals or bins. The number of data points falling in a bin indicates the the frequency of the observations in that interval. Like other lossy compression algorithms, the clustering-based algorithms controls the precision of the approximation using B bits. B bits are used to store the index of a transformed data point. Since B bits can represent 2^B different values and if the number of different change ratios in Δ, $|\Delta D_i|$, is larger than 2^B, then some of Δ must be grouped together and approximated by a representative ratio in the same group. We compute such approximation to fit all representative change ratios into a index table of size 2^B.

While one can use rather simpler methods like equal-width or log-scale binning than the clustering-based binning, they may not perform well for cases where the distribution is highly irregular. One such example would be when there are several densely packed bins and those dense bins are spread unevenly. Neither the equal-width nor the log scheme is known to be capturing such an irregularity well. Data clustering is a technique to partition data into groups with a similarity (in terms of distance) while maximizing the difference among groups. Several prior studies [10, 21, 37] also have been used clustering techniques in compression, mostly for multimedia data (images, sounds, and videos). The idea behind those techniques based on the following characteristics on multimedia data. First, the data objects are highly similar from one time frame to another, that is, small temporal changes. Second, a certain amount of information loss is acceptable in their applications. Lastly, as described in [32], a substantial data reduction

is highly desired. In many senses, many scientific simulation runs exhibit very similar behaviors as the multimedia data. This is the reason why the cluster analysis technique achieves better binning results for irregularly distributed datasets.

3.3. Error Bounding Methods

The main challenge when applying lossy compression is assessing its effect on simulation accuracy. Lossy compression is commonly used for multimedia data (photographs and videos) where errors need only to meet human perceptual requirements. Key-frames in videos periodically reset the error to zero. The scientific simulation codes solve time-dependent differential equations where errors may accumulate over time. The changes in the solution due to the use of compression may grow more rapidly if compression errors at one time step are strongly correlated with errors at the next time step. The changes due to the use of compression are thus likely to be functions of both the compression ratio and the error characteristics of the compression algorithm.

However, the fact that lossy compression may change simulation results is not necessarily a show-stopper. In many cases, a slight change in one simulation parameter also can change the simulation output. For example, scientists may choose a mesh resolution based on a trade-off between more accurate answers and the computational cost of the simulation. Computational scientists usually make these choices based on the desired accuracy of various physical quantities, not by comparing differences in per grid point basis like the mean square error between two simulations. For example, Laney *et al.* [23] have proposed to use the same integral physical quantities to assess the impact of compression.

ISABELA and *fpzip* bound errors using the accuracy metric that measures factors between the original and approximated values. For example, ISABELA uses Pearson correlation of $\geq$0.99, which implies that 99% of approximated data is within the error bound. Since this metric measures the strength and direction of the linear relationship between two, the error bound is *relative*, thus no absolute error guarantee outside the defined range. *fpzip* also allows the similar relative error bound because of its use of the non-uniform quantization [23].

In contrast to traditional lossy compression algorithms, the clustering-based scheme is designed to process data under the condition that the compressed data is guaranteed with a user-specified *absolute* error bound or a user tolerance error rate E. The value of E is usually determined based on the application domain knowledge. For each point in Δ, if $abs(\Delta) < E$, the clustering-based algorithm uses 0 as its approximation value because it already meets the user tolerance error threshold. Otherwise, it uses the clustering algorithm to learn the distribution of Δ and partitions the data in Δ based on their similarity in order to meet the user tolerance error-bound E. The compression ratio shown in Figure 1(d) was obtained while the error rate and the approximation precision are fixed at 0.1% and 8 bits, respectively. In terms of the mean error rate, all variables show less than 0.025% of error rates, guaranteeing the user-specified error rate. Similar to the clustering-based algorithm, APAX also allows the absolute error to be bounded within each APAX block. Quantization is the only source of loss in APAX, which can be tunable by the user [23].

3.4. Tradeoffs Approximation Precision and Error Rate

One obvious advantage of using lossy compression techniques is that users can tradeoff between the approximation precision and compression ratio. In order to demonstrate this, we varied the number of precision bits to see how the clustering-based lossy compression algorithm performs in terms of compression ratio and error rate. In this set of experiments, we fix the user-defined tolerable error rate at 0.1%. As one can expect, increasing the approximation bits improved the compression ratio significantly. For instance, the compression ratio increases dramatically when the number of bits changed from 8 to 9 bits, 40% to 80% while the mean error rate is increased only by 0.02%. Furthermore, if the approximation precision is 10 bits, then all data points became compressible, resulting in compression ratio of 8x with the mean error rate less than 0.05%. Other approximation schemes like log-scale or equal-with binning achieved similar results on *rlds* and other variables.

Lossy compression algorithms can perform differently depending on the amount of information loss. For example, when we vary the user tolerable error rate from 0.1% (default error bound) to 0.5%, the average compression ratio by the clustering-based scheme increased from 2.1x up to 4.7x. The mean error rate is also increased from 0.02% to 0.12% as the user tolerable error rate is increased 5 times. We however note that they are still much smaller than the user tolerable error rate. For example, we could maintain the mean error rate of 0.1% even with the user tolerable error rate of 0.4%.

3.5. Comparison

Table 2 gives a comparison of lossy compression algorithms described so far. Each scheme uses different transformation methods in order to increase compressibility, and then apply different approximation algorithms on those transformed data. As compared with Table 1, one can clearly see that lossy compression algorithms achieve much higher compression ratios; up to 8x depending on how compressible data is. Another important observation from Table 2 is that data transformation helps increase the compression ratio significantly. *fpzip* achieved only 1.29x of compression ratio because it uses only prediction mechanism based on data traversal without applying any transformation on the actual data. ISABELA and the clustering-based scheme achieved the highest compression ratio because both schemes first transform original data into a compression-friendly format.

We note that each data point (assuming 64-bit) in lossy compression algorithm is divided into two categories: compressible and incompressible. All the compressible data is represented as an approximation bit number (e.g., 8 bits) whereas the incompressible data is stored as the original bit number (i.e., 64-bit). Since the compressible portion is essentially represented as integer streams, we can further increase the compression ratio by applying one of existing lossless compression techniques like *zlib* [2], *bzlip2*, or *LZO* to the index data. As discussed in [22], indices, which are integer values, are easy to compress with standard lossless compressions, resulting in about 75%–90% of additional compression ratio.

While not extensively discussed in this paper, there are studies where data is encoded not at system-level but at application-level to tolerate faults. A recent study by Chen [9] describes an alternative technique, called Algorithm-Based Fault Tolerance (ABFT), that eschews traditional checkpointing techniques to incorporate error recovery into algorithmic design. Chen demonstrates essentially overhead-free recovery mechanisms for the Jacobi method and conju-

gate gradient descent, algorithmic error recovery mechanisms are by necessity specific to the code being run. Moreover, they are potentially vulnerable to compound or cascading failures, which periodic checkpointing would help to alleviate, even in cases where such techniques are applicable.

Table 2. Comparison of lossy compression schemes.

Scheme	Transformation Applied	Approximation Algorithm	Compression Ratio	Error bound
ISABELA [22]	sorting	apply B-spline on sorted data	up to 5x	$\geq$0.99 of correlation
fpzip [26]	not used	traverse data in a coherent order and then uses the corresponding n-dimensional (where n is the dimensionality of the data) Lorenzo predictor to predict the subsequent values. It next maps the predicted values and actual values to their integer representations, and encodes the XORd' residual between these values.	1.29x	relative
APAX [30]	not used	encodes sequential blocks of input data elements with user-selected block size between 64 and 16,384	1.33x$\sim$4x	absolute
Clustering-based	change ratios between consecutive iterations	approximate on change ratios; full checkpoint initially and when the error rate is close to user-specified bound	2.98x$\sim$8x	<0.1% absolute

4. Conclusion

This paper argues that while the traditional checkpointing continues to be a crucial mechanism to tolerate system failures in many scientific applications, it is also becoming challenging in the exascale era mainly because of limited I/O scalability and associated energy cost. This paper describes several efforts to use lossless compression on checkpoint data to relieve such overheads and shows that they are limited because of inherent randomness in scientific datasets. This paper then describes several lossy compression algorithms that radically change how checkpoint data is stored with tunable error bounding mechanisms. Hence, we predict the lossy compression to be a promising way to reduce checkpoint overheads without compromising the quality of dataset that scientific simulation operates on.

For lossy compressions to be actually deployed in the exscale computing era, several future challenges remain. The first challenge would be reducing the memory requirements to perform in-situ compression. This is especially challenging because of two facts: (1) per-core memory is expected to continue to decrease in the exascle systems, and (2) transformation techniques to improve the effectiveness of compressions typically require extra memory, making memory a scarce resource. Second, given the amount of increasing data volumes and number of CPU cores, the compression should be performed in parallel. A couple of current parallel I/O libraries support data compression, but the compression is performed individually, in order words, compression is locally optimized without knowing what others do. Lastly, compression algorithms must take advantages of several emerging techniques to be used in future exascale systems such as SSD, PCRAM, etc., as they will significantly impact future memory hierarchy.

Acknowledgement

This work is supported in part by the following grants: NSF awards CCF-1029166 and OCI-1144061, and IIS-1343639; DOE awards DE-SC0005309, DESC0005340, and DESC0007456; AFOSR award FA9550-12-1-0458. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

1. http://cmip-pcmdi.llnl.gov/cmip5/.

2. http://www.zlib.net/.

3. Ankit Agrawal, Mostofa Patwary, William Hendrix, Wei-keng Liao, and Alok Choudhary. *High Performance Big Data Clustering*, pages 192–211. IOS Press, 2013.

4. Allison H. Baker, Haiying Xu, John M. Dennis, Michael N. Levy, Doug Nychka, Sheri A. Mickelson, Jim Edwards, Mariana Vertenstein, and Al Wegener. A Methodology for Evaluating the Impact of Data Compression on Climate Simulation Data. In *Proceedings of the ACM International Symposium on High-Performance Parallel and Distributed Computing*, 2014. DOI: 10.1145/2600212.2600217.

5. Leonardo Arturo Bautista-Gomez and Franck Cappello. Improving Floating Point Compression through Binary Masks. In *Proceedings of the IEEE International Conference on Big Data*, pages 326–331, 2013. DOI: 10.1109/BigData.2013.6691591.

6. Tekin Bicer, Jian Yin, David Chiu, Gagan Agrawal, and Karen Schuchardt. Integrating Online Compression to Accelerate Large-Scale Data Analytics Applications. In *Proceedings of the IEEE 27th International Symposium on Parallel Distributed Processing*, pages 1205–1216, May 2013. DOI: 10.1109/IPDPS.2013.81.

7. Simona Boboila, Youngjae Kim, Sudharshan S. Vazhkudai, Peter Desnoyers, and Galen M. Shipman. Active Flash: Out-of-core Data Analytics on Flash Storage. In *Proceedings of the 28th Symposium on Mass Storage Systems and Technologies*, pages 1–12, 2012. DOI: 10.1109/MSST.2012.6232366.

8. Martin Burtscher and Paruj Ratanaworabhan. FPC: A High-Speed Compressor for Double-Precision Floating-Point Data. *IEEE Trans. Computers*, 58(1):18–31, 2009. DOI: 10.1109/TC.2008.131.

9. Zizhong Chen. Algorithm-based Recovery for Iterative Methods Without Checkpointing. In *Proceedings of the International Symposium on High Performance Distributed Computing*, pages 73–84, 2011. DOI: 10.1145/1996130.1996142.

10. C.-H. Chou, M.-C. Su, and E. Lai. A New Cluster Validity Measure and Its Application to Image Compression. *Pattern Anal. Appl.*, 7(2):205–220, July 2004. DOI: 10.1007/s10044-004-0218-1.

11. Jin J. Chou and Les A. Piegl. Data Reduction Using Cubic Rational B-splines. *IEEE Computer Graphics and Applications*, 12(3):60–68, May 1992. DOI: 10.1109/38.135914.

12. Thomas M. Cover and Joy Thomas. *Elements of Information Theory.* Wiley, 1991.

13. David Donofrio, Leonid Oliker, John Shalf, Michael F. Wehner, Chris Rowen, Jens Krueger, Shoaib Kamil, and Marghoob Mohiyuddin. Energy-Efficient Computing for Extreme-Scale Science. *IEEE Computer*, 42(11):62–71, 2009. DOI: 10.1109/MC.2009.353.

14. Jason Duell. The Design and Implementation of Berkeley Labs linux Checkpoint/Restart. Technical report, Lawrence Berkeley National Laboratory, 2003.

15. Michael Frazier. *An Introduction to Wavelets through Linear Algebra*. Undergraduate texts in mathematics. Springer, 1999.

16. Nathanael Hübbe, Al Wegener, JulianMartin Kunkel, Yi Ling, and Thomas Ludwig. Evaluating Lossy Compression on Climate Data. In JulianMartin Kunkel, Thomas Ludwig, and HansWerner Meuer, editors, *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, volume 7905 of *Lecture Notes in Computer Science*, pages 343–356, 2013.

17. Dewan Ibtesham, Dorian C Arnold, Patrick G Bridges, Kurt B Ferreira, and Ron Brightwell. On the Viability of Compression for Reducing the Overheads of Checkpoint/Restart-Based Fault Tolerance. In *Proceedings of the International Conference on Parallel Processing*, pages 148–157, 2012. DOI: 10.1109/ICPP.2012.45.

18. Tanzima Zerin Islam, Kathryn Mohror, Saurabh Bagchi, Adam Moody, Bronis R. de Supinski, and Rudolf Eigenmann. McrEngine: A Scalable Checkpointing System Using Data-aware Aggregation and Compression. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 17:1–17:11, 2012.

19. John Jenkins, Isha Arkatkar, Sriram Lakshminarasimhan, David A. Boyuka II, Eric R. Schendel, Neil Shah, Stéphane Ethier, Choong-Seock Chang, Jackie Chen, Hemanth Kolla, Scott Klasky, Robert B. Ross, and Nagiza F. Samatova. ALACRITY: Analytics-Driven Lossless Data Compression for Rapid In-Situ Indexing, Storing, and Querying. *Transactions on Large-Scale Data- and Knowledge-Centered Systems X - Special Issue on Database- and Expert-Systems Applications*, 10:95–114, 2013. DOI: 10.1007/978-3-642-41221-9_4.

20. Ruoming Jin, Anjan Goswami, and Gagan Agrawal. Fast and Exact Out-of-core and Distributed K-means Clustering. *Knowl. Inf. Syst.*, 10(1):17–40, 2006. DOI: 10.1007/s10115-005-0210-0.

21. Nicolaos B. Karayiannis and Pin-I Pai. Fuzzy Vector Quantization Algorithms and Their Application In image Compression. *IEEE Transactions on Image Processing*, 4:1193–1201, 1995. DOI: 10.1109/83.413164.

22. Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F. Samatova. Compressing the Incompressible with ISABELA: In-situ Reduction of Spatio-temporal Data. In *Proceedings of the 17th International Conference on Parallel Processing*, pages 366–379, 2011. DOI: 10.1007/978-3-642-23400-2_34.

23. Daniel Laney, Steven Langer, Christopher Weber, Peter Lindstrom, and Al Wegener. Assessing the Effects of Data Compression in Simulations Using Physically Motivated Metrics. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 76:1–76:12, 2013. DOI: 10.1145/2503210.2503283.

24. Didier Le Gall. MPEG: A Video Compression Standard for Multimedia Applications. *Commun. ACM*, 34(4):46–58, April 1991. DOI: 10.1145/103085.103090.

25. Wei-keng Liao. Parallel K-Means Data Clustering. http://www.eecs.northwestern.edu/~wkliao/Kmeans/, 2005.

26. Peter Lindstrom and Martin Isenburg. Fast and Efficient Compression of Floating-Point Data. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):1245–1250, Sept 2006. DOI: 10.1109/TVCG.2006.143.

27. Jay F. Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. Flexible IO and Integration for Scientific Codes Through the Adaptable IO System (ADIOS). In *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments*, pages 15–24, 2008. DOI: 10.1145/1383529.1383533.

28. Khalid Sayood. *Introduction to Data Compression (2nd Ed.)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

29. Eric R. Schendel, Ye Jin, Neil Shah, Jackie Chen, Choong-Seock Chang, Seung-Hoe Ku, Stphane Ethier, Scott Klasky, Robert Latham, Robert B. Ross, and Nagiza F. Samatova. ISOBAR Preconditioner for Effective and High-throughput Lossless Data Compression. In *Proceedings of the 28th IEEE International Conference on Data Engineering*, 2012. DOI: 10.1109/ICDE.2012.114.

30. Eric R. Schendel, Saurabh V. Pendse, John Jenkins, David A. Boyuka, II, Zhenhuan Gong, Sriram Lakshminarasimhan, Qing Liu, Hemanth Kolla, Jackie Chen, Scott Klasky, Robert Ross, and Nagiza F. Samatova. ISOBAR Hybrid Compression-I/O Interleaving for Large-scale Parallel I/O Optimization. In *Proceedings of the International ACM Symposium on High Performance Parallel and Distributed Computing*, June 2012. DOI: 10.1145/2287076.2287086.

31. Neil Shah, Eric R. Schendel, Sriram Lakshminarasimhan, Saurabh V. Pendse, Terry Rogers, and Nagiza F. Samatova. Improving I/O Throughput with PRIMACY: Preconditioning ID-Mapper for Compressing Incompressibility. In *Proceedings of the IEEE International Conference on Cluster Computing*, September 2012. DOI: 10.1109/CLUSTER.2012.16.

32. Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining.* Addison Wesley, May 2005.

33. Karl E. Taylor, Ronald J. Stouffer, and Gerald A. Meehl. An Overview of CMIP5 and the Experiment Design. *Bull. Amer. Meteor. Soc.*, 93(4):485–498, October 2012.

34. The HDF Group. http://www.hdfgroup.org/HDF5/.

35. Josep Torrellas. Architectures for Extreme-Scale Computing. *Computer*, 42(11):28–35, November 2009. DOI: 10.1109/MC.2009.341.

36. Benjamin Welton, Dries Kimpe, Jason Cope, Christina M. Patrick, Kamil Iskra, and Robert Ross. Improving I/O Forwarding Throughput with Data Compression. In *Proceedings of the IEEE International Conference on Cluster Computing*, pages 438–445, 2013. DOI: 10.1109/CLUSTER.2011.80.

37. Yueting Zhuang, Yong Rui, Thomas S. Huang, and Sharad Mehrotra. Adaptive Key Frame Extraction using Unsupervised Clustering. In *Proceedings of the International Conference on Image Processing*, pages 866–870, 1998. DOI: 10.1109/ICIP.1998.723655.