NUMARCK: Machine Learning Algorithm for Resiliency and Checkpointing

Zhengzhang Chen, Seung Woo Son, William Hendrix, Ankit Agrawal, Wei-keng Liao, and Alok Choudhary
Electrical Engineering and Computer Science Department
Northwestern University
Evanston, IL 60208
Email: {zzc472, sson, whendrix, ankitag, wkliao, choudhar}@eecs.northwestern.edu

***Abstract*—Data checkpointing is an important fault tolerance technique in High Performance Computing (HPC) systems. As the HPC systems move towards exascale, the storage space and time costs of checkpointing threaten to overwhelm not only the simulation but also the post-simulation data analysis. One common practice to address this problem is to apply compression algorithms to reduce the data size. However, 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. This paper exploits the fact that in many scientific applications, the relative changes in data values from one simulation iteration to the next are not very significantly different from each other. Thus, capturing the distribution of relative changes in data instead of storing the data itself allows us to incorporate the temporal dimension of the data and learn the evolving distribution of the changes. We show that an order of magnitude data reduction becomes achievable within guaranteed user-defined error bounds for each data point.**

We propose NUMARCK, *N*orthwestern *U*niversity *M*achine learning *A*lgorithm for *R*esiliency and *C*hec*K*pointing, that makes use of the emerging distributions of data changes between consecutive simulation iterations and encodes them into an indexing space that can be concisely represented. We evaluate NUMARCK using two production scientific simulations, FLASH and CMIP5, and demonstrate a superior performance in terms of compression ratio and compression accuracy. More importantly, our algorithm allows users to specify the maximum tolerable error on a per point basis, while compressing the data by an order of magnitude.

I. Introduction

The challenges of extreme scale computing systems [24], [9] exist across multiple dimensions including architecture, energy constraints, memory scaling, limited I/O, scalability of software and applications. It is clear that the larger the simulations using extreme-scale systems, (1) the greater the need for effective resiliency, (2) at a faster pace, and (3) within the constrains of limited (relatively) storage space, deeper and more complex memory hierarchies, minimization of data movement (particularly external to the systems) due to energy, I/O and other constraints. The traditional models of storing raw and uncompressed data as checkpoints will frequently become cost prohibitive. On the other hand, it will remain necessary to be able to store the states of simulations for resiliency and restart purposes. Lossy compression [7], [10], on the other hand, can help somewhat in reducing data size, but the error rates are not easy to bound, and in large scale simulations significant deviation from values can impact the outcome of the simulation. Thus brute-force solutions that don't consider multiple dimensions of the problem to scale resilience via checkpointing are unlikely to satisfy the constraints posed by such a large system. Some rethinking is required that considers the problem top-down and within multiple constraints.

A simulation calculates values on points (nodes, particles, etc.) in a discretized space based on mathematical models and proceeds along the temporal dimension. One can think of this as the system transitioning from one state to the next. There are data values (variable values) in each state and there is a transition (call it change) from one state to the next. A typical checkpoint attempts to store a state regardless of transition or change. For argument sake, let's assume that only one point's value changes from one time-step to the next out of a hundred million points. Traditional techniques would still store 100 million points for both time steps regardless of this fact. What if we stored the change or relative change? In this case, one needs to only note the fact that one point changed and one can rebuild the next state based on this fact. Of course, the previous example is trivial. The following questions arise. (1) How do we calculate change at scale? (2) How do we represent change at scale so that the representation reduces the data size to be stored by an order of magnitude or more, thereby addressing the I/O problem and storage size problem? (3) How do we learn the patterns of change? (4) How do we perform the above tasks while minimizing data movement (more computations locally for learning patterns of change)? (5) How do we bound any errors from approximations? And, (6) How do we engineer scalable software for storing, replaying, and restarting simulations?

What if we could guarantee point-wise error bounds at levels never imagined before, defined by a user as a design goal, while reducing the space to store data by an order of magnitude or more, while performing lot of the computations locally (to reduce data movement) and taking advantage of data reduction by potentially being able to use node-local non-volatile storage? This paper presents a set of techniques to achieve the above goals based on algorithms and machine learning techniques to learn temporal change patterns, along with a data representation to capture changes along with algorithms and engineering solutions for storing the data in which a user can indicate the maximum point-wide tolerable error.

We consider that checkpointing will continue to be a very important component of resiliency, particularly if the data can be significantly reduced with guaranteed point-wise error bounds that a user can control. Furthermore, the error tolerance can be specified by the user as a parameter. For example, in

978-1-4799-5500-8/14 $31.00 © 2014 IEEE
DOI 10.1109/SC.2014.65

our approach, a user can indicate that maximum tolerable error per point should be less than 0.1%, and our techniques will meet that bound, while further reducing the mean error by two orders of magnitude below the point-wise tolerable error. Our approach also reduces data movement, thereby potentially reducing the power usage which is expected to be dominated by data movement.

To address this problem, we propose NUMARCK (pronounced "Nu-Mark"), the *NU M*achine learning *A*lgorithm for *R*esiliency and *C*hec*K*pointing. The fundamental idea of NUMARCK is to learn emerging distributions by capturing temporal changes, thereby performing *in-situ* checkpoint approximation and reducing the data significantly (an order of magnitude) with guarantees of error-bounds for every point. More specifically, (1) similar to forward predictive coding in video compression, we first compute the relative change in data values between two consecutive timestamps or iterations; (2) then a machine learning-based data approximation algorithm is designed to determine the distribution of change and encode it with low cost in space and minimal data movement (mostly in place). Each data point is represented by its corresponding group's approximation value. If the difference ratio between the approximated value and true value is larger than a user defined error threshold, the exact value is stored; and (3) when some fault interrupts the systems or the simulations are aborted, our framework can restart the simulation by using the compressed data. We successfully apply our algorithm to climate CMIP5 (Coupled Model Intercomparison Project) simulation and the FLASH simulation. Our experimental results show that our algorithm outperforms existing lossy compression methods such as B-Splines.

The remainder of this paper is organized as follows. Our design of our methodology is described in Section II. Section III presents our experimental setup and evaluation results. We discuss related work in Section IV. Finally, we summarize our findings and discuss future work in Section V.

II. Method

A. Overview

The intuition behind our NUMARCK method stems from the following three observations. First, scientific simulation data, like climate CMIP5 rlus (Surface Upwelling Longwave Radiation) data, are considered as one type of high entropy data [8], [18]. Such data often exhibits randomness without any distinct repetitive patterns in one single timestamp or iteration (see Fig. 1 (A) or (B)). Thus, traditional *lossless* compression approaches [4], [17] cannot achieve appreciable data reduction. Second, in many scientific applications, relative changes in data values from one simulation iteration to the next are often not very significant from the perspective of the distribution of the change itself. For example, more than 75% of climate rlus data remains unchanged or only changes with a percentage less than 0.5% (see Fig. 1 (C) and (D)). Third, unlike observational data, many scientific applications can tolerate some error-bounded loss in their simulation data accuracy. Thus, *lossy* compression methods can offer some attractive features for simulation data reduction. However, the effectiveness of lossy compression heavily depends on the domain knowledge to select the right compression algorithms,

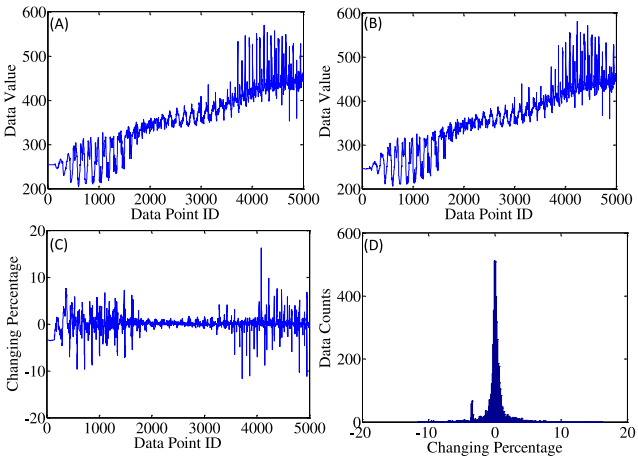


Fig. 1: A slice of climate rlus simulation data: (A) original data of iteration 1; (B) original data of iteration 2; (C) the changing percentage of data values between two iterations; (D) the distribution of relative data change between two iterations.

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 user's specified error rate.

In order to achieve the exascale goals, we must use scalable thinking, which is orthogonal to current approaches. *What if we could learn temporal patterns and evolving distributions of data that also involves time dimension with the goal of data reduction? Furthermore, what if we could bound the error to some user defined parameter?* The checkpoint approximation framework for exascale scientific simulations consists of several main continuous procedures, which are illustrated in algorithm 1. In this paper, we propose a temporal data approximation approach for scientific simulation checkpointing. An overview of our NUMARCK approach vs. traditional checkpointing mechanism is given in Fig. 2. The fundamental idea is to learn emerging distributions by capturing temporal changes, thereby performing in-situ checkpoint approximation, reducing the data significantly (an order of magnitude) with guarantees of per point error-bounds. Our approach consists of three stages: (1) forward predictive coding by calculating the relative change in data values between two consecutive timestamps or iterations; (2) data approximation by determining distribution of change and encoding it into another space that can be concisely represented with low cost in space and minimal data movement (mostly in place); and (3) simulation restart by using the compressed data during the system restart/failure stage.

Scientific simulations use predominantly double-precision floating-point variables, so the remainder of the paper will focus on data approximation/compression for these variables, though our method can be applied to floating point numbers of different precision as well.

B. Forward Predictive Coding

As a first step, we transform the data by computing the relative change in data values from one iteration to the next.

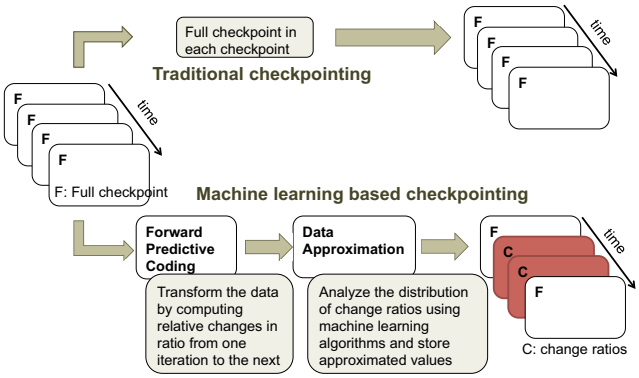


Fig. 2: Overview of traditional checkpoint vs. NUMARCK.

Algorithm 1: NUMARCK: Checkpoint Approximation Framework for Large Scale Scientific Simulations

Input:

D: scientific simulation data
E: user tolerance error threshold
B: the approximation precision
: (i.e., the number of bytes to store each data point)
I : the number of checkpoint iterations
J : the number of data points in each iterations
S : the beginning iteration of restart

Output:

D': approximation data
ϵ : restart data

```
/* The first checkpoint is compressed
   by using lossless compression or
   storing as it is;                    */
```

$C_0 \leftarrow Lossless(D_0)$;
for $\forall i \leq I$ **do**
  `/* Forward predictive coding;  */`
  $\triangle D_i \leftarrow \frac{D_i - D_{i-1}}{D_{i-1}}$;
  for $\forall j \leq J$ **do**
    if $abs(D_{i,j}) < E$ **then**
      $D_{i,j} \leftarrow D_{i-1}$;
    end
    else
      $D'_{i,j} \leftarrow approximation(\triangle D_{i,j}, B, E)$;
    end
  end
end
for $\forall S \leq s \leq I$ **do**
  `/* Restart using the approximation data;  */`
  $\epsilon_s \leftarrow restart(D'_S, S)$;
end

Our data transformation idea shares the same spirit with video compression algorithms especially MPEG [16] which stores the differences between successive frames using temporal compression. To exploit temporal redundancy between frames, MPEG codes the remaining frames using two prediction techniques: (a) forward predictive coding, which codes the actual frame with reference to a past frame; and (b) bidirectional prediction, which uses a past and a future frame to code the current frame [16]. But because our checkpointing framework is designed to enable the computing systems to recover the data after some fault interrupts the systems or when the simulations are aborted, only forward predictive coding is employed in our work.

And instead of computing the difference between two successive frames, we calculate the relative change (called "change ratio") as follows:

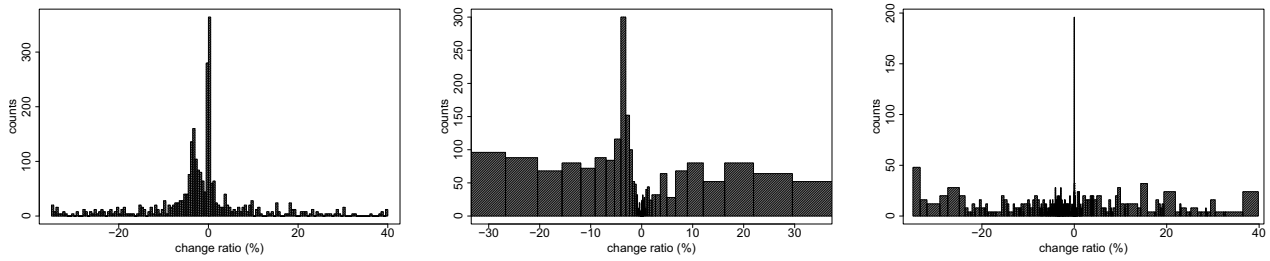
$$\Delta D_{i,j} = \frac{D_{i,j} - D_{i-1,j}}{D_{i-1,j}}, \tag{1}$$

where $D_{i,j}$ and $D_{i-1,j}$ is the value of the jth data point in iteration i and iteration $i-1$, respectively, and $i \geq 1$. Note that $D_{i-1,j}$ cannot be zero. If $D_{i-1,j}$ is zero, $D_{i,j}$ will be stored as it is. And D_0 is the first checkpoint, which will be stored as the exact values.

One question one might ask is what is so special about "change" in data values between two snapshots as opposed to the values in individual snapshots? Let us take an example of 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 10 percent change. Similarly, data points with the same change percentage can be indexed by one number. Our idea of considering the data changes along temporal domain transforms the data where repeated patterns are rare in individual snapshots 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. We will address this challenge in next subsection.

C. Data Approximation by Learning Distribution

Once the change ratios of all data points have been calculated, using machine learning techniques, we first calculate the distributions of changes and then transform them into an indexed space to achieve the goals of maximal data reduction. In contrast to traditional lossy compression algorithms, our data compression algorithm is designed to process data under the condition that the compressed data is guaranteed with a user-specified error bound or a user tolerance error rate E. The value of E is usually determined based on the application domain knowledge. In addition to E, we provide another user parameter for controlling the precision of the approximation, B the number of bits used to store the index of a transformed data point. Since B bits can represent up to a maximum of 2^B different values and if the number of different change ratios in ΔD_i, $|\Delta D_i|$, is larger than 2^B, then some of ΔD_i must be grouped together and approximated by a representative



(a) Equal-width binning. (b) Log-scale binning. (c) Clustering-based approximation.

Fig. 3: The histogram of the 255 bins for the dens FLASH data between iteration 32 and 33 using three different approximation strategies.

ratio in the same group. We compute such approximation to fit all representative change ratios into a index table of size 2^B. For each $\Delta D_{i,j}$, if $abs(\Delta D_{i,j}) < E$, we use 0 as its approximation value because it already meets the user tolerance error threshold. For $abs(\Delta D_{i,j}) >= E$, in order to meet the user tolerance error-bound E, we design several strategies to learn the distribution of ΔD_i and partition the data in ΔD_i based on their similarity.

1) Equal-width Binning: Histogram is a popular method to learn the data distribution (as shown in Fig. 1 (D)). 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. Given k bins, we can only assign data points to a maximum of $k = 2^B - 1$ bins, as 0 being used to store all $\Delta D_{i,j}$ with $abs(\Delta D_{i,j}) < E$.

The equal-width binning method partitions the change ratios in ΔD_i into $2^B - 1$ bins of equal width. Each ratio $\Delta D_{i,j}$ is then represented by the bin ID and approximated by the bin center. If the difference between the original value and the approximated one is larger than user-specified error rate E, we store it as is (i.e., uncompressed). One obvious disadvantage of this equal-width binning strategy is that it is highly dependent on the range of all ratios in ΔD_i, as the bin width is calculated by dividing the range by the number of bins. If the bin width W is smaller than $2 * E$, then ΔD_i can be perfectly compressed by B bits, meaning all compressed ratios are within the user-specified error threshold. However, the maximum range of the change ratios that can be covered by the equal-width binning strategy is $2 * E * (2^B - 1)$. If $W > 2 * E$, and majority of the data points are distributed at the edges of the bins, equal-width binning will produce a poor compression result.

2) Log-scale Binning: To address the limitation of equal-width binning when data covers a large range of values, we propose a log-scale binning method, which assigns data points into bins based on their e-based logarithm values. Similar to the equal-width binning, log-scale binning also partitions the change ratios into $2^B - 1$ bins, but the bin widths are in log scale. This strategy allows to cover larger range of change ratios, and more finer (narrower) bins can be assigned to smaller changes and coarser (wider) bins for larger changes. An example is shown in Fig. 3b.

3) Clustering-based Approximation: Although both equal-width and log-scale binning methods can depict the probability distribution of the data values, they do not perform well for highly irregularly distributed data. For instance, when there are multiple dense areas in the histogram and the dense areas are spread unevenly, neither methods can capture distribution with such characteristics. Data clustering, also known as vector quantization, is a technique to partition data into groups of similar data objects while maximizing the difference between groups. Clustering has also been used in compression for multimedia data, such as image, sound, and video data, [14], [6], [26], as they exhibit the following common characteristics: (1) the data objects are highly similar to one another, (2) some loss of information is acceptable, and (3) a substantial reduction in the data size is desired [22]. If viewed along the temporal dimension, most of the scientific simulation data have the similar behaviors as the multimedia data. Therefore, we employ a cluster analysis technique to obtain locally dense areas that can help derive a better binning result.

Our approach first applies the parallel K-means clustering algorithm[1] [1], [13] on the temporal change ratios ΔD_i to get $2^B - 1$ clusters. One intrinsic limitation in K-means clustering is that the choice of initial clustering centroids has been proved to influence significantly the performance of the algorithm and quality of the results. To overcome this limitation, we initialize the cluster centroids for K-means with prior-knowledge from the equal-width histogram to achieve more reliable segmentation results. Then a table is created that consists of the prototypes of $2^B - 1$ clusters. A cluster prototype contains information of the number of change ratios fall into this bin, the centroid of the bin, and other metadata. Each prototype is indexed in the table with an integer of length B bits. Each data $\Delta D_{i,j}$ is represented by the index of the cluster it belongs. The cluster centroid becomes the approximation value for all the data contained in the corresponding cluster.

D. Restart With Compressed Data

During restart, NUMARCK first reads the latest uncompressed, complete full checkpoint (denoted as D_0) as the data in all intermediate checkpoint files are in approximated change

[1]The software package for parallel K-means clustering developed by our group can be found in http://users.eecs.northwestern.edu/~wkliao/Kmeans/index.html.

ratios between two consecutive checkpoints. NUMARCK then reads the intermediate checkpoint files and applies each of them to the full checkpoint data in order to build the restart file.

Our restart mechanism rebuilds each data point using the following method:

$$\epsilon_{i,j} = \begin{cases} D_{i,j}, & \text{if } \zeta_{i,j} = 0 \\ D'_{i-1,j} * (1 + \triangle D'_{i,j}), & \text{otherwise,} \end{cases}$$

where $D_{i,j}$ is the value for j-th data points in iteration i, $D'_{i,j}$ and $D'_{i-1,j}$ are the jth data points in an approximated form at iteration i and $i-1$, respectively, $\epsilon_{i,j}$ is a re-constructed restart data, and $\zeta_{i,j}$ is the index of data $D_{i,j}$ to indicate if $D_{i,j}$ is compressible or not. If $\zeta_{i,j}$ is equal to zero, it means that $D_{i,j}$ is incompressible thus stored as an exact value (i.e., $D_{i,j}$). Otherwise, $\zeta_{i,j}$ is equal to 1, which means $D_{i,j}$ is compressible.

Note that, $\epsilon_{i,j}$ can be either an exact value (i.e., the same as in the full checkpoint data) or approximated value. We repeat this step for all the intermediate checkpoint files.

Since our restart phase is built from the approximated change ratios, the reconstructed restart data points inherently include some error. Depending on how far a restart point from the last full checkpoint, different amount of error can be accumulated depending on error rate exhibited at each checkpoint step.

III. Results

In this section, we first describe the datasets and evaluation metrics used in our experiments, then evaluate NUMARCK algorithm on two production scientific simulations, FLASH and CMIP5. We focus on answering the following four questions:

1) What is the performance of our algorithm using the three different approximation techniques? See Section III-C.
2) What is the effect of our approximation parameters like user tolerant error-bound and approximation precision? See Section III-D and Section III-E.
3) What is the performance of our algorithm compared to other benchmark lossy compression algorithms? See Section III-F.
4) Will the checkpointing restart mechanism be affected by our approximation results or not? See Section III-G.

A. Datasets

To evaluate our checkpoint compression and restart performance, we use a collection of simulation checkpoint data generated by the FLASH and climate CMIP5 simulation.

The FLASH [11] is a block-structured adaptive mesh hydrodynamic code that solves the compressible Euler equations on a block structured adaptive mesh and incorporates the necessary physics to describe the environment, including the equation of state, reaction network, and diffusion. The problem domain is divided into blocks distributed among a number of MPI processes. A block is a three-dimensional array with an additional 4 elements as guard cells in each dimension on both sides to hold information from its neighbors. There are 24 data variables per array element, and about 80 blocks on each MPI process. A variation in block numbers per MPI process is used to generate a slightly unbalanced I/O load. Because of the fixed number of blocks for each process, an increase in the number of processes linearly increases the total number of blocks. We set the block size to be 16×16, which corresponds to about 64 MB of data per process. There are 24 variables, each of which is written using collective write calls. During checkpointing, FLASH writes one checkpoint file and two plot files for visualization, which contain centered and corner data. Since we focus on checkpoint/restart, in our experiments, we evaluated the checkpoint files only. Among all 24 variables, only 10 variables are written to checkpoint files: *dens, eint, ener, gamc, game, pres, temp, velx, vely, velz*.

CMIP (Coupled Model Intercomparison Project) is supported by WCRP (World Climate Research Program). The CMIP5 (CMIP Phase 5) experiment design [23] has been finalized with the following suites of experiments: (1) Decadal Hindcasts and Predictions simulations, (2) "long-term" simulations, (3) "atmosphere-only" simulations for especially computationally-demanding models. Out of the dozens variables available in CMIP5, we randomly chose five including mrsos (Moisture in Upper Portion of Soil Column), mrro (Total Runoff), mc (Convective Mass Flux), rlds (Surface Downwelling Longwave Radiation), rlus (Surface Upwelling Longwave Radiation), and abs550aer (Ambient Aerosol Absorption Optical Thickness at 550 nm). The resolution for these data is 2.5° by 2°. mc is a monthly simulation data, while other four are daily data.

B. Evaluation Metrics

We utilize several metrics to evaluate the performances: mean error rate, incompressible ratio, and compression ratio.

Mean error rate is used to measure the accuracy of our data approximation algorithm. It is the average difference between the approximated change ratio and the real change ratio across all data points for each iteration.

We also used the maximum error rate, which is the maximum difference between the approximated change ratio and the real change ratio across all data points for each iteration.

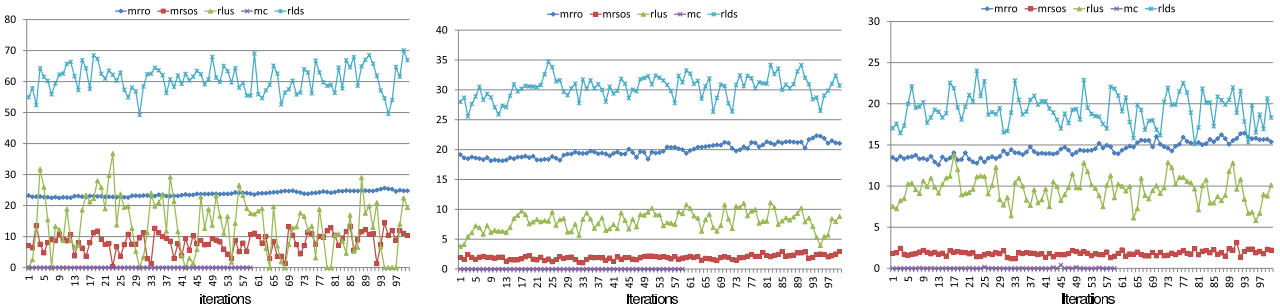
Incompressible ratio γ is the fraction of the data needed to be stored as exact values in one iteration.

The compression ratio R for data D of size $|D|$ reduced to size $|D^{'}|$ is defined as:

$$R = \frac{|D| - |D^{'}|}{|D|} \times 100\%. \quad (2)$$

With incompressible ratio γ, the bit number B used to store the approximation index, our NUMARCK algorithm's compression ratio R can be defined as:

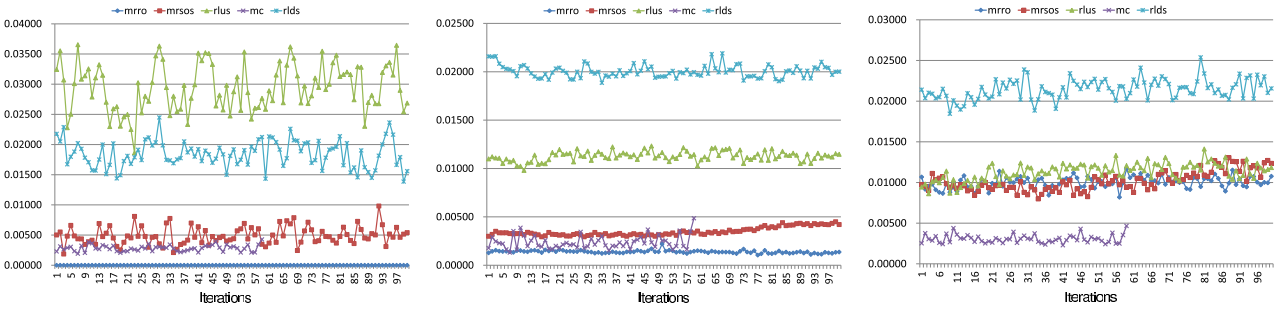
$$R = \frac{|D| - ((1-\gamma) * \frac{B}{64} + \gamma * |D| + (2^B - 1) * 64)}{|D|} \quad (3)$$



(a) Incompressible ratio (%) for the equal-width binning.

(b) Incompressible ratio (%) for the log-scale binning.

(c) Incompressible ratio (%) for the clustering-based approximation.

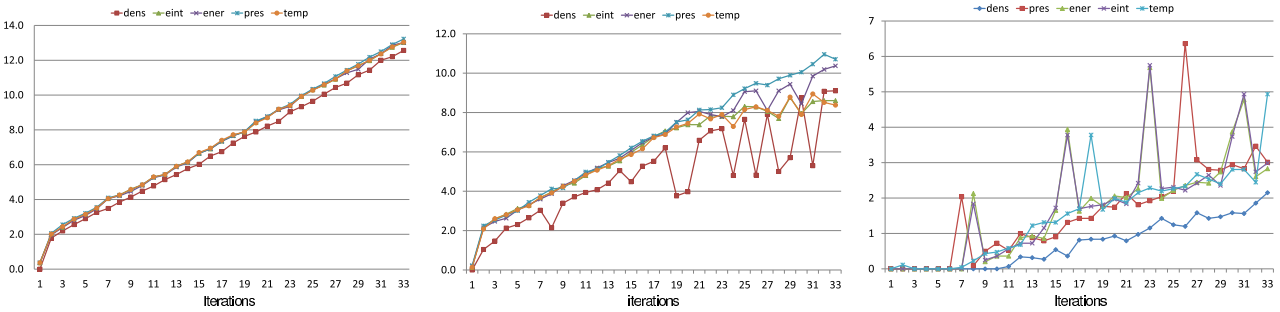


(d) Mean error rate (%) for the equal-width binning.

(e) Mean error rate (%) for the log-scale binning.

(f) Mean error rate (%) for the clustering-based approximation.

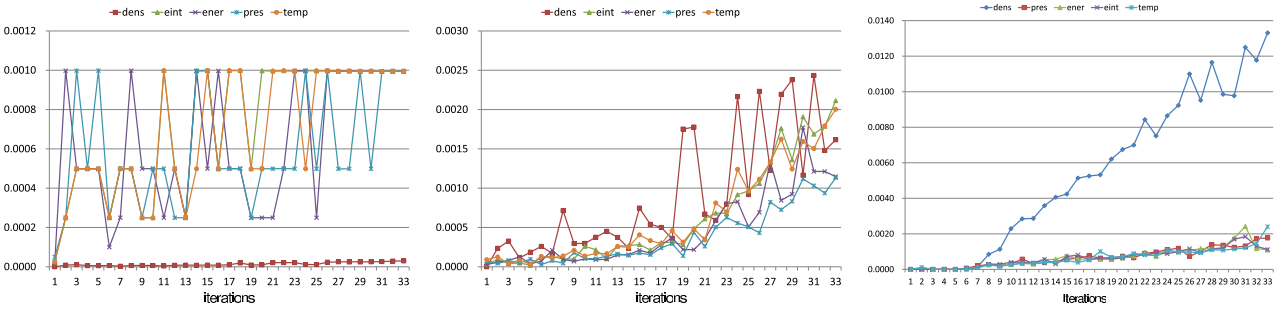
Fig. 4: NUMARCK's performance on CMIP5 simulation data. The point-wise user specified error rate $E = 0.1\%$.



(a) Incompressible ratio (%) for the equal-width binning.

(b) Incompressible ratio (%) for the log-scale binning.

(c) Incompressible ratio (%) for the clustering-based approximation.



(d) Mean error rate (%) for the equal-width binning.

(e) Mean error rate (%) for the log-scale binning.

(f) Mean error rate (%) for the clustering-based approximation.

Fig. 5: NUMARCK's performance on FLASH simulation data. The point-wise user specified error rate $E = 0.1\%$.

where $(1-\gamma)*\frac{B}{64}$ is the storage requirement for the index of the compressed data, $\gamma * |D|$ is for the incompressible data, and $(2^B-1)*64$ is for the approximation values/representive values of the 2^B-1 bins or clusters.

We can further use a lossless compression technique like FPC [4] on our compressed data to achieve higher compression ratio. But since how to design a lossless algorithm to compress the data approximation result is out of the scope of this work, we will not include lossless compression result in this paper.

C. Approximation Performance of Different Strategies

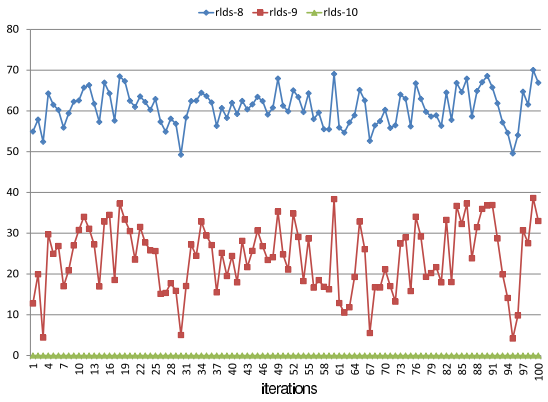
We tested the three data approximation strategies, including equal-width binning, log-scale, and clustering-based method, on FLASH and CMIP5 simulation data with ten different variables. We set the user tolerance error threshold to be 0.1% and the approximation precision to be 8 bits. Fig. 4 and 5 show that the clustering-based strategy achieves the best performance on all ten datasets in terms of incompressible ratio, and log-scale method outperforms the equal-width binning. We also observe that CMIP5 data are more challenging than the FLASH data for data approximation task. For example, clustering-based method achieves less than 7% incompressible ratio on all FLASH data, while it only gets maximum 25% incompressible ratio in CMIP5 data. In terms of mean error rate, three approximation strategies achieve comparable results with less than 0.025% on all datasets.

D. Effect of Different Data Approximation Precision

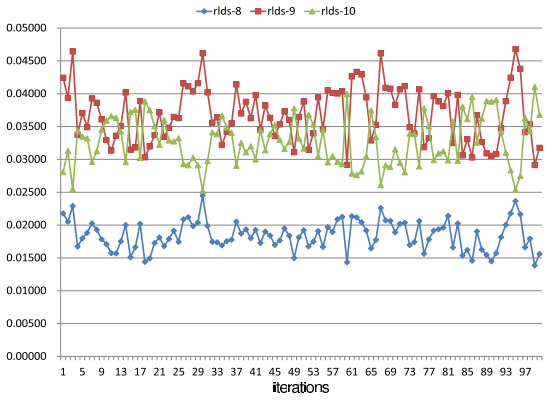
So far, we used 8 bits to store the approximation results. In this experiment, we want to demonstrate how NUMARCK's performance gets affected by the approximation precision. Here, the simple equal-width binning strategy is used, and CMIP rlds data is chosen. The user tolerance error threshold is set to be 0.1%. As shown in Fig. 6, if the approximation precision is increased to 9 bits instead of 8 bits, the average incompressible ratio decreases dramatically from 60% to 20%, and the average compression ratio increases by more than 30%, while the mean error rate only increases by less 0.02%. Even better, if the approximation precision is increased to 10 bits, all data points now become compressible and the average compression ratio increases to nearly 85%, while all mean error rate are below than 0.05% which is only an half of the user tolerance error threshold. Log-scale binning strategy and clustering-based strategy achieved similar trends on the rlds data and other simulation data.

E. Effect of Different Error Rates

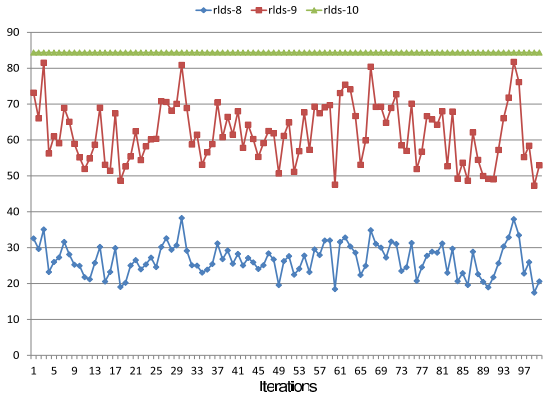
So far, we fixed the user tolerance error threshold to be 0.1%. In this experiment, we want to demonstrate how our NUMARCK algorithm's performance gets affected by the user tolerance error threshold. Here, the clustering-based approximation strategy is used, and one of the most challenging simulation data–CMIP abs550aer data is chosen. The user tolerance error threshold is set to be 0.1%. As shown in Fig. 7, with the user tolerance error is increasing from 0.1% to 0.5%, the average incompressible ratio keeps decreasing from more than 40% to less than 10%, and the average compression ratio keeps increasing from less than 50% to more than 80%. Although the mean error rate increases from 0.02% to 0.12%,



(a) Incompressible ratio (%).



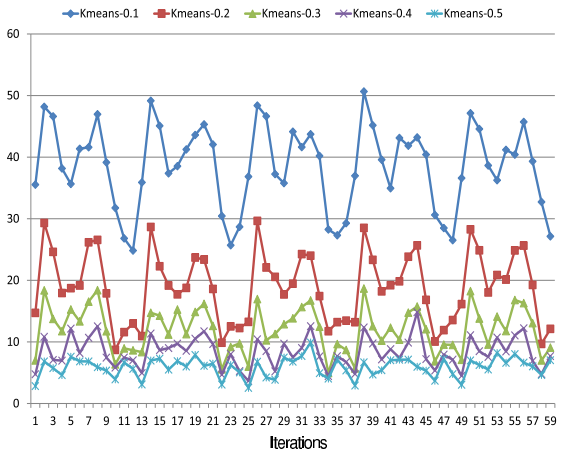
(b) Mean error rate (%).



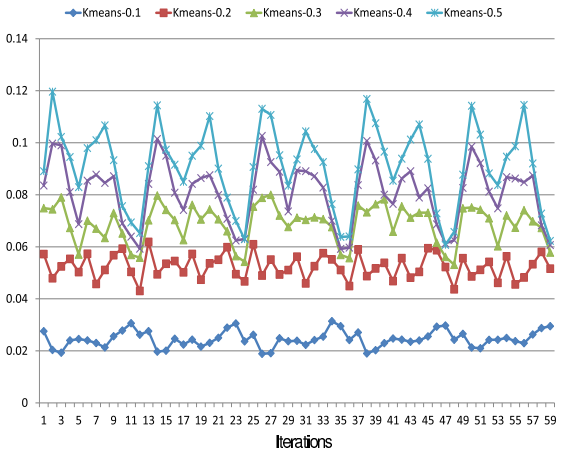
(c) Compression ratio (%).

Fig. 6: NUMARCK's performance with different data approximation precisions on *rlds* data for 100 iterations when the equal-width binning is used.

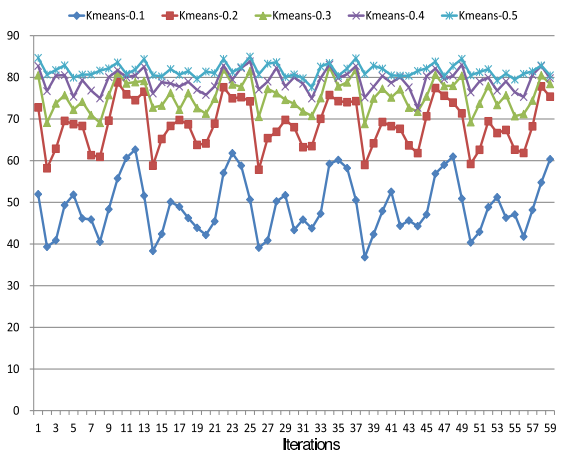
all mean error rate is way smaller than the user tolerance error. For example, for the fixed the user tolerance error 0.4%, the average mean error rate is still less than 0.1%.



(a) Incompressible ratio (%).



(b) Mean error rate (%).



(c) Compression ratio (%).

Fig. 7: NUMARCK's performance with different user tolerance error threshold on *abs550aer* data for 60 iterations when the clustering-based approximation is used.

F. Comparison with Lossy Compression Algorithms

In this section, we compare the performance of NUMARCK with B-Splines [7] and ISABELA [15] algorithms on 10 scientific datasets from CMIP5 and FLASH simulations. The compression accuracy is measured by using Pearson's correction coefficient (ρ) and root mean square error (ξ) between the original data vector $D = (d_1, d_2, ..., d_n)$ and decompressed data vector $D^{'} = (d_1^{'}, d_2^{'}, ..., d_n^{'})$ defined as follows:

$$\xi = \sqrt{\frac{\sum_{i=1}^{n}(d_i - d_i^{'})^2)}{n}} \quad (4)$$

TABLE I: Compression ratio comparison on ten simulation data.

	B-Splines	ISABELA	NUMARCK
rlus	20±0.000	80.078±0.000	**81.776±0.014**
mrsos	20±0.000	80.078±0.000	**81.947±0.012**
mrro	20±0.000	**80.078±0.000**	77.587±0.000
rlds	20±0.000	80.078±0.000	**80.756±0.097**
mc	20±0.000	80.078±0.000	**82.002±0.000**
dens	20±0.000	75.781±0.000	**87.476±0.000**
pres	20±0.000	75.781±0.000	**86.725±0.009**
temp	20±0.000	75.781±0.000	**86.876±0.004**
ener	20±0.000	75.781±0.000	**86.849±0.004**
eint	20±0.000	75.781±0.000	**86.856±0.002**

We compare the averages of compression ratio R, Pearson's correction coefficient (ρ) and root mean square error (ξ) across 50 iterations. The compression ratio of ISABELA algorithm is decided by the window size W_0 and the number of B-splines constants P_I [15], while the compression ratio of regular B-splines algorithm is solely dependent on the number of constants P_S, which requires $P_S * 64$ bits to store the compressed data. And the compression ratio of our NUMARCK algorithm is defined by Eq.3.

For fair comparison, we assign the same number of bits for storing the approximation index in ISABELA and NUMARCK, that is the window size W_0 and B. More specifically, we use $W_0 = 2^9 = 512$ and $B = 9$ for compressing the CMIP5 simulation data including rlus, mrsos, mrro, rlds and mc, while we use $W_0 = 2^8 = 256$ and $B = 8$ for compressing the FLASH simulation data because previous experimental results showed that FLASH data is easier to compress than CMIP5 data. The P_I is fixed to be 30 as suggested in paper [15]. The user tolerant error-bound $E = 0.5\%$ and clustering-based approximation strategy are used in NUMARCK for all datasets. For B-Splines algorithm, because it requires $P_S \sim n$, where n is the dimension of the data vector of one iteration, to provide accurate lossy compression, we assign $P_S = 0.8 * n$ for all datasets.

The comparison results can be seen in Table I and II. Our NUMARCK algorithm got higher compression ratio than ISABELA and B-Splines in 9 out of 10 datasets. Table II shows that out of 10 datasets, 9 datasets exhibit $\rho = 0.999$ with our algorithm. The ξ values for B-Splines are consistently an order of magnitude higher than ISABELA and NUMARCK. NUMARCK outperforms ISABELA on all datasets with

TABLE II: Compression accuracy comparison on ten simulation data.

	ρ			ξ		
	B-Splines	ISABELA	NUMARCK	B-Splines	ISABELA	NUMARCK
rlus	**0.999± 0.000**	**0.999±0.000**	**0.999±0.000**	2.677±0.010	0.703±0.002	**0.524±0.034**
mrsos	0.987±0.000	**0.989±0.000**	0.987± 0.000	1.874±0.001	0.397±0.001	**0.365±0.000**
mrro	0.968±0.001	0.967±0.001	**0.999± 0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**
rlds	0.999±0.001	**0.999±0.000**	**0.999±0.000**	3.213±0.025	0.638±0.003	**0.567±0.007**
mc	**0.999±0.000**	**0.999±0.000**	**0.999±0.000**	207.561±1.876	215.727±1.791	**188.488±159.827**
dens	0.989±0.000	0.998±0.000	**0.999±0.000**	0.048± 0.000	0.014±0.000	**0.000±0.000**
pres	0.994±0.000	0.998±0.000	**0.999±0.000**	0.950± 0.022	0.287±0.004	**0.008±0.000**
temp	0.993±0.000	0.997±0.000	**0.999±0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**
ener	0.993±0.000	0.998±0.000	**0.999±0.000**	15.178 ±0.113	4.833±0.097	**0.176 ±0.004**
eint	0.993±0.000	0.998±0.000	**0.999±0.000**	15.030±0.053	4.891±0.085	**0.171±0.004**

smaller average root mean square errors. Note that the larger variance (i.e., 159.827) of ξ, NUMARCK got, on *mrsos* is due to the big range of ξ values ($[155.196, 214.864]$) over the 50 iterations. But even the maximum ξ value of NUMARCK on *mrsos* is still smaller than the mean ξ value (i.e., 215.727) of ISABELA.

G. Checkpoint/Restart

This experiment shows how the application restart mechanism works using our checkpoint files. Fig. 8 shows the mean and maximum error rate when the FLASH was rerun using reconstructed restart file when the equal-width, log-scale and clustering-based binning strategy, respectively. In order to show the impact of accumulated error rates through intermediate checkpoint files (in approximated value), we varied the reconstruction points up to 4 from the full complete checkpoint files. In order to see the impact of restarting simulation at different checkpoint steps, we ran all restart runs up to 8 consecutive checkpoints. We measured the mean and maximum error, both are accumulated, throughout all restart experiments.

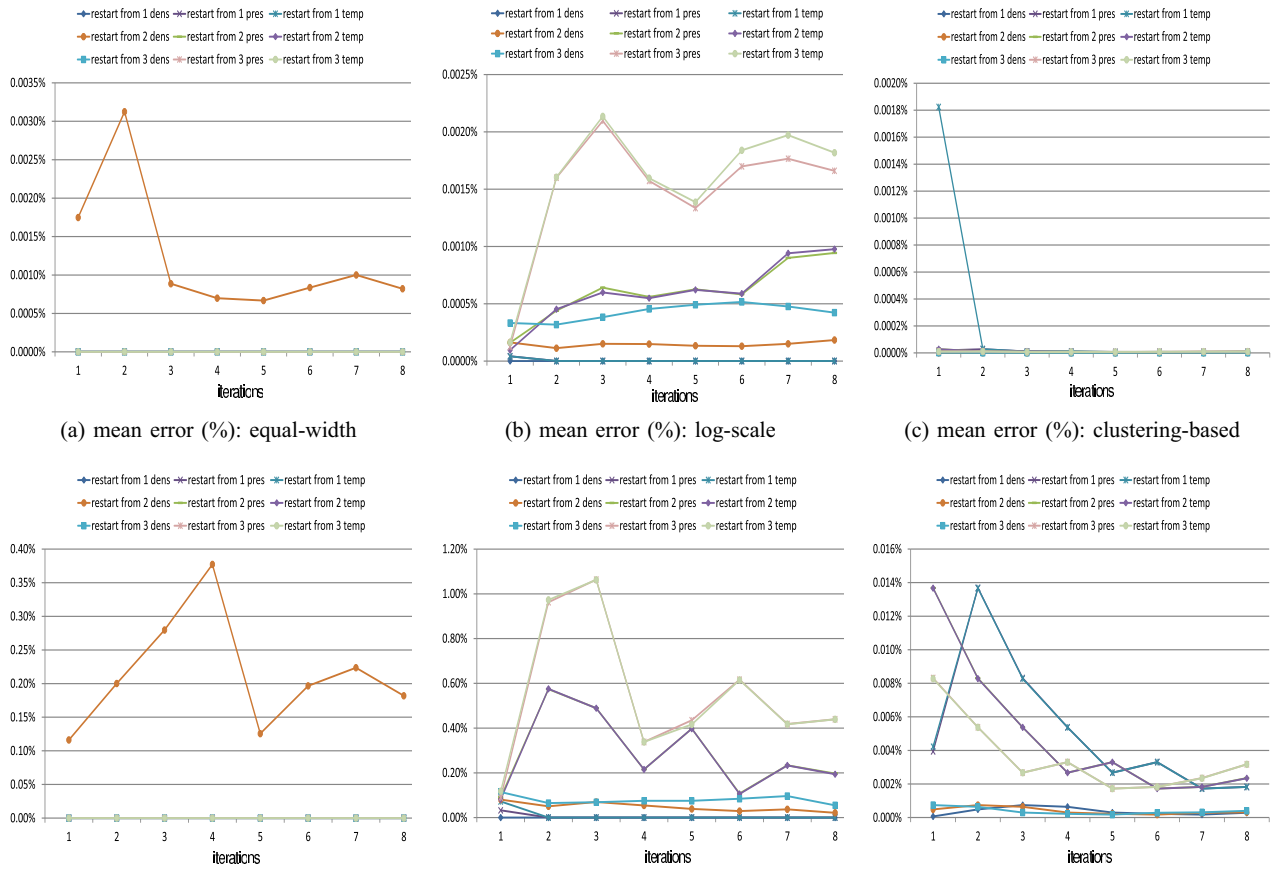
We make several observations. First of all, we observe that FLASH can run successfully using the restart files that are reconstructed from approximated values. This is a promising result because simulation codes can run with restart files that have a certain error bounds. We also observe that two variables (*pres* and *temp*) showed very similar behaviors because the computation applied to both is actually the same. Therefore, the error rate are affected the same way for both. In general, although there are some variations throughout checkpoints, the mean error rate are very small; most are far below the threshold we used (0.1%). As shown in the previous section, *dens* is easy to compress regardless of whichever binning strategy is used, so it showed minimal variations in error rates during restarted checkpoints. Another important thing we observed is that the farther restart points from the full checkpoint, the higher maximum error rate. This is expected because higher restart point means more accumulated error. The last important observation from these results is that in certain checkpoints, the restart files made out of the log binning showed higher error rates than the equal-width binning. This is because there is a tradeoff between the compression ratio and the error tolerated between two binning strategies. Remember that we showed that the equal-width binning strategy achieved lower compression ratio than the logging binning. This means that more data points are stored as exact values (i.e., incompressible) in the equal-width binning, thereby introducing less error rates during restart phases. Lastly, we observe that the clustering-based binning strategy gives the lowest maximum error rates. Also, only the clustering-based strategy does not exceed our specified error bound (0.1%). Combined with the highest compression ratio we achieved, we conclude that bins identified by our clustering-based method best represent the distribution of ratio changes in terms of data reduction and error rates.

IV. Related Work

Given the looming challenge of I/O scalability in breaking the barrier to exascale, it is not surprising that several other approaches have already been proposed to overcome the challenges of I/O at exascale. However, in the context of checkpointing, much of this work has focused on the application of lossless compression techniques, as many researchers assume that restart data must replicate the previous state exactly. As such, we primarily contrast our work against lossless techniques for reducing checkpointing overhead. In reality, however, simulation parameters are often calibrated using data that are themselves subject to measurement error or other inaccuracies. Critically, very small deviations ($< 1\%$) in restart fidelity are unlikely to obscure the actionable insights from scientific simulations, allowing for much higher performance gains than much of the existing work in this area.

Welton *et al.* [25] describe IOFSL, an I/O middleware that invisibly integrates compression algorithms with standard POSIX and MPI I/O. They evaluate their framework on using widely-available compression algorithms on synthetic and real datasets. 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. Incorporating other algorithms, though, such as the one described in this paper, could enable seamless integration with existing scientific simulation codes.

Islam *et al.* [12] present mcrEngine, a software framework that interleaves compatible data between checkpoint files before applying a compression algorithm selected by data type (double vs. float vs. other) and compressing the output of this algorithm with a general purpose compression algorithm. The authors tested mcrEngine on five different codes that exhibited this type of compaitibility between checkpoint files and compressed the checkpoint data by a factor of 1.2–5. While



(a) mean error (%): equal-width (b) mean error (%): log-scale (c) mean error (%): clustering-based

(d) max error (%): equal-width (e) max error (%): log-scale (f) max error (%): clustering-based

Fig. 8: Mean and maximum error rate when FLASH is restarted with reconstructed checkpoint files using different binning strategies. The reconstructed restart files at different checkpoint (2, 3, and 4), and the FLASH simulation continued 8 more checkpoints (iterations) from restarting. Note that the error rates in (a), (c), and (d) are very close to zero except one case.

this checkpoint-level similarity may not be applicable to every simulation code, the technique of recombining this data before compression has clear strengths.

Bicer *et al.* [3] propose a novel compression algorithm for climate data, CC, that takes advantage of the spatial and temporal locality inherent in climate data to accelerate storage and retrieval for climate applications. 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 "cancel out" to form easily compressed datasets. Their method can also be extended to lossy compression, though they do not present any results showing compression greater than 65%.

Schendel *et al.* propose ISOBAR [20], [19], [21], an I/O framework that incorporates data compression to accelerate I/O at scale. While their technique also divides the data into compressible and incompressible segments, ISOBAR uses lossless compression to reduce data size, resulting in lower performance than lossy techniques.

Bautista-Gomez and Capello [2] 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 these independently. Bautista-Gomez and Capello, however, "pre-condition" the data by applying a bitwise (`xor`) mask to the data in order to reduce its entropy before compression. They present results for a number of scientific datasets, achieving a maximum of around 40% compression.

While not targeted to work on checkpoint data, other work on lossy compression could potentially be applied in this space to a similar effect. For example, Lakshminarasimhan *et al.* [15] described ISABELA, a compression technique based on applying B-splines to sorted data. By preconditioning the data, ISABELA was able to achieve high compression on data that was previously regarded as "incompressible," while losing minimal data fidelity (≥ 0.99 correlation with original data). Lakshminarasimhan *et al.* did not consider applying ISABELA to checkpoint data because they assume checkpoint data do not permit approximation.

As an alternative technique, Chen [5] describes Algorithm-Based Fault Tolerance (ABFT), a technique that eschews

traditional checkpointing techniques to incorporate error recovery into algorithmic design. Chen demonstrates essentially overhead-free recovery mechanisms for the Jacobi method and conjugate 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.

V. Conclusion

Many scientific applications rely on a checkpoint/restart mechanism to tolerate system failures, which is becoming increasingly challenging for extreme-scale computing systems because of limited I/O, storage capacity, power requirements and other costs. This paper has demonstrated that capturing the distribution of relative changes in data instead of storing data itself allows us to bring in the temporal dimension of the checkpoint data and learn the evolving distribution of the changes. We show that an order of magnitude data reduction becomes achievable with user-defined and guaranteed error bounds by transforming the learned distributions into another space. We applied several machine learning-based binning mechanisms to store those relative changes approximately within a guaranteed user-specified error rate. Our evaluation with the checkpoint data from two scientific simulations (CMIP5 and FLASH) indicates that our mechanism allows for a very high compression ratio while guaranteeing user-specified error rate. We also showed that our approach achieves better compression ratio with the fixed error rate or higher accuracy with the fixed compression ratio than existing lossy compression techniques. Finally, we showed that how restart files can be reconstructed from approximated checkpoint files, and that the FLASH simulation can actually restart successfully while maintaining the error rate within user-specified bounds.

What we present in this paper is just a first step towards a scalable resiliency solution including checkpoint/restart mechanism for extreme scale systems. Design of functions, local computations, minimizing data movements and other aspects are a part of future work. NUMARCK's mechanisms in learning the evolving data distributions can also enable understanding anomalies at scale, thereby potentially identifying erroneous calculations due to soft errors or hardware errors. Furthermore, adaptation of these techniques can help enable scalable in-situ analysis as well as determining dynamic checkpointing frequency based on how evolving distributions change.

Acknowledgment

This work is supported in part by the following grants: NSF awards CCF-1409601, CCF-1029166, and ACI-1144061, and IIS-1343639; DOE awards DE-SC0005309, DESC0005340, and DESC0007456; AFOSR award FA9550-12-1-0458.

References

[1] A. Agrawal, M. Patwary, W. Hendrix, W.-k. Liao, and A. Choudhary, *High Performance Big Data Clustering*. IOS Press, 2013, pp. 192–211.

[2] L. A. Bautista-Gomez and F. Cappello, "Improving floating point compression through binary masks," in *Proceedings of the IEEE International Conference on Big Data*, 2013, pp. 326–331.

[3] T. Bicer, J. Yin, D. Chiu, G. Agrawal, and K. Schuchardt, "Integrating Online Compression to Accelerate Large-Scale Data Analytics Applications," in *IEEE 27th International Symposium on Parallel Distributed Processing*, May 2013, pp. 1205–1216.

[4] M. Burtscher and P. Ratanaworabhan, "FPC: A high-speed compressor for double-precision floating-point data." *IEEE Trans. Computers*, vol. 58, no. 1, pp. 18–31, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/tc/tc58.html#BurtscherR09

[5] Z. Chen, "Algorithm-based Recovery for Iterative Methods Without Checkpointing," in *Proceedings of the International Symposium on High Performance Distributed Computing*, 2011, pp. 73–84.

[6] C.-H. Chou, M.-C. Su, and E. Lai, "A New Cluster Validity Measure and Its Application to Image Compression," *Pattern Anal. Appl.*, vol. 7, no. 2, pp. 205–220, July 2004.

[7] J. J. Chou and L. A. Piegl, "Data Reduction Using Cubic Rational B-splines," *IEEE Computer Graphics and Applications*, vol. 12, no. 3, pp. 60–68, May 1992.

[8] T. M. Cover and J. Thomas, *Elements of Information Theory*. Wiley, 1991.

[9] D. Donofrio, L. Oliker, J. Shalf, M. F. Wehner, C. Rowen, J. Krueger, S. Kamil, and M. Mohiyuddin, "Energy-Efficient Computing for Extreme-Scale Science," *IEEE Computer*, vol. 42, no. 11, pp. 62–71, 2009.

[10] M. Frazier, *An introduction to wavelets through linear algebra*, ser. Undergraduate texts in mathematics. Springer, 1999.

[11] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo, "FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes," *The Astrophysical Journal Supplement Series*, vol. 131, no. 1, p. 273, 2000.

[12] T. Z. Islam, K. Mohror, S. Bagchi, A. Moody, B. R. de Supinski, and R. 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*, 2012, pp. 17:1–17:11.

[13] R. Jin, A. Goswami, and G. Agrawal, "Fast and Exact Out-of-core and Distributed K-means Clustering," *Knowl. Inf. Syst.*, vol. 10, no. 1, pp. 17–40, 2006.

[14] N. B. Karayiannis and P.-I. Pai, "Fuzzy Vector Quantization Algorithms and Their Application In image Compression," *IEEE Transactions on Image Processing*, vol. 4, pp. 1193–1201, 1995.

[15] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova, "Compressing the Incompressible with ISABELA: In-situ Reduction of Spatio-temporal Data," in *Proceedings of the 17th International Conference on Parallel Processing*, 2011, pp. 366–379.

[16] D. Le Gall, "MPEG: A Video Compression Standard for Multimedia Applications," *Commun. ACM*, vol. 34, no. 4, pp. 46–58, April 1991.

[17] P. Ratanaworabhan, J. Ke, and M. Burtscher, "Fast Lossless Compression of Scientific Floating-Point Data," in *DCC*, 2006, pp. 133–142.

[18] K. Sayood, *Introduction to Data Compression (2nd Ed.)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000.

[19] E. Schendel, Y. Jin, N. Shah, J. Chen, C. Chang, S.-H. Ku, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. Samatova, "ISOBAR Preconditioner for Effective and High-throughput Lossless Data Compression," in *Proceedings of the 28th IEEE International Conference on Data Engineering*, 2012.

[20] E. Schendel, S. Pendse, J. Jenkins, D. B. II, Z. Gong, S. Lakshminarasimhan, Q. Liu, S. Klasky, R. Ross, and N. 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.

[21] N. Shah, E. R. Schendel, S. Lakshminarasimhan, S. V. Pendse, T. Rogers, and N. 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.

[22] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison Wesley, May 2005.

[23] K. E. Taylor, R. J. Stouffer, and G. A. Meehl, "An Overview of CMIP5 and the Experiment Design," *Bull. Amer. Meteor. Soc.*, vol. 93, no. 4, pp. 485–498, Oct. 2012. [Online]. Available: http://dx.doi.org/10.1175/bams-d-11-00094.1

[24] J. Torrellas, "Architectures for Extreme-Scale Computing," *Computer*, vol. 42, no. 11, pp. 28–35, November 2009.

[25] B. Welton, D. Kimpe, J. Cope, C. M. Patrick, K. Iskra, and R. Ross, "Improving I/O Forwarding Throughput with Data Compression," in *Proceedings of the IEEE International Conference on Cluster Computing*, 2013, pp. 438–445.

[26] Y. Zhuang, Y. Rui, T. S. Huang, and S. Mehrotra, "Adaptive Key Frame Extraction using Unsupervised Clustering," in *Proceedings of the International Conference on Image Processing*, 1998, pp. 866–870.