Recovering Detectable Uncorrectable Errors via Spatial Data Prediction

Kristen Guernsey
Holcombe Department of Electrical and Computer Engineering - Clemson University
Clemson, SC, United States of America
kguerns@clemson.edu

Sarah Placke
Holcombe Department of Electrical and Computer Engineering - Clemson University
Clemson, SC, United States of America
splacke@clemson.edu

Alexandra Poulos
Holcombe Department of Electrical and Computer Engineering - Clemson University
Clemson, SC, United States of America
alpoulo@clemson.edu

Jon C. Calhoun
Holcombe Department of Electrical and Computer Engineering - Clemson University
Clemson, SC, United States of America
jonccal@clemson.edu

ABSTRACT

High-performance computing applications are central to advancement in many fields of science and engineering. Central to this advancement is the supposed reliability of the HPC system. However, as system size grows and hardware components run with near-threshold voltages, transient upset events become more likely. Many works have explored the problem of correcting data corruption; however, recovery is often left to checkpoint-restart or application-specific techniques. Recovering from a checkpoint incurs overhead due to reading a checkpoint and recomputing lost work. Allowing the application to recover just the corrupted data enables faster and more efficient recovery. This paper explores using spatial similarities to recover detectable uncorrectable errors. We explore several reconstruction methods and evaluate their effectiveness at recovering corrupted entries in data arrays. Results show that the Lorenzo 1-Layer prediction method yields the best results, with over half of its reconstructions having less than 1% relative error across all applications.

CCS CONCEPTS

• **Hardware → Error detection and error correction**; **Failure recovery, maintenance and self-repair**; • **Software and its engineering → Software fault tolerance**.

KEYWORDS

Detectable Uncorrectable Errors, High-performance Computing, Forward Recovery, Data prediction

ACM Reference Format:
Kristen Guernsey, Sarah Placke, Alexandra Poulos, and Jon C. Calhoun. 2023. Recovering Detectable Uncorrectable Errors via Spatial Data Prediction. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3624062.3624120

SC-W 2023, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0785-8/23/11.
https://doi.org/10.1145/3624062.3624120

1 INTRODUCTION

High-performance computing (HPC) applications enable scientific discovery across many disciplines. However, as systems use more complex components that are run at lower voltages to reduce power consumption, the rate of faults has been shown to increase [17, 28]. Faults, which often manifest as bit-flips in data paths, registers, or memory storage cells, cause errors, which are classified as follows. Detectable and correctable errors (DCE) are detected by the system and the corrupted value is recovered, allowing the application to continue running. Detectable but uncorrectable errors (DUE) are detected, but the system is unable to recover the corrupted value. This typically results in an unexpected application crash that forces the process to recover from a previous state, or checkpoint [25]. The remaining possibility is that the error goes undetected, wherein it generates deviations in the process's state not sufficient to crash the application. These deviations are referred to as silent data corruption (SDC), and if not detected can result in the application computing on erroneous data. Over time, this corruption can propagate throughout a simulation and compromise the fidelity of the application's final output [7].

Several techniques have been developed to detect the presence of SDC. Traditional approaches rely on application redundancy in space or time, but come at a high cost [21]. To reduce this overhead, detection techniques which verify physical invariants in a simulation - such as conservation laws [24] - or use data analytics to detect outliers [3, 9, 27] have been developed. Other approaches rely on modifying the algorithm or data structures to be more resilient [11, 16, 18]. The traditional method to recover from SDC and DUE reinitializes the state of the program via a checkpoint [23, 25]. Alternatively, forward recovery seeks to rebuild the corrupted state in the application by using the non-corrupted state [2, 12, 13, 16, 27]. Central to the success of forward recovery techniques is knowing where corruption exists in the application. Knowing which element(s) in a data array are corrupted allows for lower-cost localized recovery. For example, if it is known that only one entry of an array is corrupted, then we only need to restore a single entry.

Prior approaches leverage spatial prediction for fine-grained, localized recovery for individual data values [12, 13, 26, 27] but suffer from a lack of context due to their generic low-level design. For example, [12] and [13] work at the instruction level and only have limited context of the application, function, and data. Incorporating higher-level information from the application such as dimensionality enables more accurate data reconstruction methods. Knowing the dimensionality of the data enables determining the spatial neighbors of a corrupted data element. Data dimensionality and spatial prediction have already been shown to be effective tools to detect SDC [3, 9], but these techniques have not been used to reconstruct corrupted data entries. Other work uses spatial similarity via linear interpolation to recover outliers [27]. Spatial data prediction forms the heart of the SZ lossy data compressors [10, 20, 29] enabling compression ratios ranging from 10–100×.

Because of its effectiveness of predicting data for HPC lossy data compression and its ability to detect SDC, we explore spatial data prediction for localized DUE recovery. Thus, we are able to convert DUE into DCE. This paper makes the following contributions:

- investigates low-cost spatial prediction techniques to recover from DUEs;
- demonstrates the relationship between data set smoothness and reconstruction accuracy; and
- shows Lorenzo 1-Layer Prediction is an accurate reconstruction method for DUEs with at least half of its corrections having less than 1% relative error across all applications.

The remainder of this paper is outlined as follows: Section 2 provides background on and motivation for the use of spatial prediction to recover from detectable uncorrectable errors. Next, in Section 3 we describe the spatial prediction algorithms and how we use them to recover from DUEs. Section 4 presents our analysis of our methods and states implications of our findings. We describe related works in the field in Section 5. Finally, we conclude in Section 6.

2 BACKGROUND AND MOTIVATION

HPC simulations use numerical methods that leverage spatially contiguous values to advance a simulation's state from time-step to time-step. For example, Figure 1 shows a standard 5-point stencil. During a simulation, values at the stencil points for time-step t_k (blue circles) are used to compute the data value at the center of the stencil at time-step t_{k+1} (red circle). Thus, using spatial values to recover corrupted data entries should yield similar results. If the exact stencil pattern is used, it may reconstruct the data exactly.

Consider the iterative stencil based Jacobi method for solving the 2D Heat diffusion problem (Equation 1). The temperature solution, $T(t, x, y)$, at point (x, y) and at time $t + 1$ is given by:

$$T(t+1,x,y) = 0.25\,(T(t,x-1,y) + T(t,x+1,y) + T(t,x,y-1) + T(t,x,y+1))) \quad (1)$$

The temperature at point x, y is computed solely from its neighbors. Thus, if that entry was corrupted, and we reconstructed the entry via averaging based on this 2D stencil, we reconstruct the data via the Jacobi method. Therefore, combining spatial values around the corrupted entry is an attractive solution for low-cost data recovery.

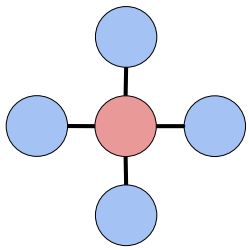

Figure 1: A 5-point stencil in two dimensions. The values of the blue points at time-step t_k are used to compute the value of the red center point at time-step t_{k+1}.

Prior work exploits the spatial and temporal smoothness of HPC simulations to detect SDC solely based on data behavior, flagging outlier values as possible corruptions when they fall outside an expected range [3, 9]. Other work seeks to recover from DUEs in memory by attempting to reconstruct or replace the erroneous datum [13, 26, 27]. In [12], Fang et al. simply ignore that an uncorrectable error has occurred and allow the application to continue running, relying on the algorithm's intrinsic robustness to localized perturbations to mask the effect of a corruption. This "compute through errors" approach is leveraged in linear solver resilience to recover from minor deviations [6, 8].

This work differs from prior reconstruction work [12, 13, 26, 27] by leveraging higher-level application information. Having knowledge of the dimensionality of a memory allocation allows us to employ multi-dimensional spatial prediction and regression functions to attempt to reconstruct the data. Moreover, we dynamically determine the best reconstruction method to use in the region around the corruption to improve reconstruction accuracy.

3 SPATIAL PREDICTION BASED RECOVERY

The de-facto method of recovering corrupted data in HPC at the software level is checkpoint-restart. In checkpoint-restart, the application periodically saves enough of its state such that when a failure occurs or errors are detected, the application rolls back its execution to the time when the checkpoint is taken. Thus, voiding potentially sizable amounts of computation, and on average, is equal to half of the time of the checkpointing period [30]. This expensive recovery must occur even if only a small subset of data is corrupted. Much lower overhead is possible if the location of the corruption is localized and low-cost methods are used to reconstruct just the corrupted portions of data.

3.1 Error Detection and Localization

Localized recovery from a detectable uncorrectable error (DUE) or silent data corruption (SDC) requires knowledge of the location of the corruption in memory. This work focuses on the pin-point recovery of errors that exist inside elements of data buffers in main-memory and/or cache. While it is possible for DUEs to corrupt multiple data elements in an array, this paper is limited to the corruption of a single element. The recovery design relies on two techniques to detect the presence of corruption.

The first method uses the machine-check architecture (MCA) for DUEs in memory [15]. The machine-check architecture is an error reporting mechanism on Intel based processors such that when a machine error — e.g., error correcting code errors, parity errors, system bus errors, TLB errors — is detected, the error type and location is recorded in certain registers before a machine-check exception (MCE) is generated. Catching the exception and inspecting the registers allows for pinpointing the exact location where the error occurs. For memory errors, we know the exact memory address. We relate the memory address to a data array to perform low-cost recovery.

The second method we utilize to detect the presence of data corruption is point-wise data analytic based inspectors [3, 9]. The algorithms leverage the fact that HPC applications that simulate the evolution of physical phenomena through time exhibit temporal and spatial smoothness in the data. Thus, these data analytic approaches compute an expectation on what each data point in an array should be or a range of plausibility. If the value is outside the range, we flag this element as corruption and initiate recovery.

3.2 Protecting Memory Allocations

Our detection methods identify the memory address or element of an array that is corrupted. However, we have only a limited view of what data is spatially contiguous to the corrupted value. For MCA, we only have a memory address. Without knowing the alignment of the data at the address and the size of each element, we are not able to apply the recovery algorithms in Section 3.4. Moreover, without knowing the dimensions of the data, the data analytic detectors only know the linear predecessor and successor elements, limiting which recovery algorithms are available.

In order to determine the data type of the corrupted memory location and its spatial neighbors, we construct a registry table of all important memory allocations[1]. When registering a memory region, the user provides the base address of the array, the data type, and the dimensionality. Optionally, the user can leverage their domain-specific knowledge to assist in the recovery process by specifying a specific recovery method to use when reconstructing the data. If not specified, we determine the locally optimal method.

To register a memory region, we use an approach similar to [3, 4], where memory allocations are registered using a function call (see Algorithm 10), where the 3D array `d3d` is recovered by any method and the 2D array `d2d` is recovered by the Lorenzo method. The prior works integrate data analytics based detectors into the Fault Tolerance Interface library [5]. FTI is an easy to use multi-level checkpointing library that makes efficient use of the complex memory hierarchy in HPC systems. We extend the FTI library to allow for forward-recovery internally after SDC is encountered, providing a means to continue application execution without having to terminate the simulation[2]. We define additional interfaces to `FTI_Protect` that record the dimensionality and the recovery method inside the *FTIT_dataset* structure that FTI uses to store all the metadata for a protected dataset. When DUE is detected in an element of a protected array, we recover using the recorded recovery method. By integrating these recovery mechanisms with the FTI library, there is no additional overhead incurred outside the cost of recovering the corrupted value. Thus, our recovery cost is much lower than the overhead of rolling back an application to a prior state via checkpoint-restart.

[1]In our design, the important memory allocations are the same ones that are checkpointed, but our design is not limited to just the checkpointed variables.
[2]Our approach is applicable to other multi-level checkpointing libraries such as SCR [22] and VeloC [25].

Algorithm 1: Array registration.

```
1  Function main()
2      d3d ← init_3d(N,N,N);
3      d2d ← init_2d(N,N);
4      FTI_Protect(0,&d3d, 3D, dtype, N, N, N, RECOVER_ANY);
5      FTI_Protect(1, &d2d, 2D, dtype, N, N, RECOVER_LORENZO);
6      for t ← 0 to T do
7          update_vars(d3d, d2d);
8          FTI_sdccheck() ;              /* Detects & corrects SDC */
9      end
10     return;
```

3.3 Recovering Corrupted Data

Once corruption has been detected, we initiate recovery by relating the offending memory address to a data allocation using our table of registered memory allocations. If the memory address is not registered, we recover using checkpoint-restart. After locating the correct memory allocation, we use the listed recovery method to reconstruct the data, before continuing execution. If `RECOVER_ANY` is listed, we initiate a localized auto-tuning to select the method that is locally optimal before returning control to the main application.

3.4 Spatial Prediction Algorithms

Each of the following reconstruction algorithms approximates the value of a corrupted datum based on some combination of spatial neighbors. Each of the methods selected vary in the amount of data required for the reconstruction and the accuracy of the reconstruction. We evaluate the effectiveness of these algorithms in Section 4. To ease the discussion of each method, Table 1 defines common notation we use when presenting the algorithms. In particular, we denote the reconstruction function as f_p and the data array as V.

Symbol	Description
d	Number of Dimensions
i	Slowest changing corrupted index of two-dimensional data set
j	Fastest changing corrupted index of two-dimensional data set
$f_p(i,j)$	Prediction Value at index (i,j)
$V(i,j)$	Existing Value at index (i,j)

Table 1: Notation.

3.4.1 Zero. Multiple prior works investigate the use of replacing corrupted information with zero [12, 13], $f_p(i,j) = 0$. Although low-cost, this method can suffer from large reconstruction error if the range of the data is large and not centered about zero.

3.4.2 Random. Prior work on SDC recovery investigates the use of a random value to replace the corrupted entry [13]. We implement this method to compare with prior work in the area. However, instead of a fully random value — i.e., unbounded — we calculate a random value within the range of the data array V as

$$f_p(i,j) = min(V) + R(max(V) - min(V)), \tag{2}$$

where $R \in [0, 1]$.

3.4.3 Average. Spatial averaging is used to compute the solution in stencil-based Jacobi relaxation (see Section 2). Similar prior work also shows that HPC data exhibits spatial smoothness [10, 13]. This means that nearby values are often close in magnitude to each other, and averaging neighboring values is a low-cost and attractive method for reconstruction. However, the accuracy of this method is dependent on the smoothness of the neighboring values. The average recovery method considers immediate adjacent values across all dimensions. Thus, this method works best if spatial smoothness is exhibited across dimensions.

3.4.4 Curve Fitting. In [10], the authors utilize a curve-fitting model to predict values when encoding a 1D data stream. These methods consist of preceding-neighbor fitting, linear-curve fitting, and quadratic-curve fitting. Here, we use these methods to predict the value of a corrupted data point. For data sets that are multidimensional, we linearize the data as in [10].

Preceding-neighbor fitting simply assigns the preceding value to the corrupted data point, i.e., $f_p(i) = V(i-1)$. The linear-curve fitting method fits a linear line through two consecutive data points to estimate the corrupted data value such that $f_p(i) = 2V(i-1) - V(i-2)$. Quadratic curve-fitting assumes that the corrupted data point lies on a quadratic curve, and uses three consecutive values to recover the data $f_p(i) = 3V(i-1) - 3V(i-2) + V(i-3)$.

3.4.5 Lorenzo Prediction. The Lorenzo prediction [14] methods have been popularized and extended in HPC community as they form a fundamental component of the SZ lossy compressor [29]. The original Lorenzo predictor [14] extends the two-dimensional parallelogram predictor to up to four dimensions, and considers only a single layer of values within each dimension surrounding the datum being predicted. SZ's version extends the original Lorenzo predictor to up to 4-layers. While an increased number of layers considers more spatial values for prediction, it can also bring uncorrelated or noisy data into the calculation. With layer-customization, SZ achieves more than 2× the compression ratio of the original SZ due to more accurately predicting data. For the details of the Lorenzo equations, we refer the reader to the equations in [29].

Typical implementations of Lorenzo prediction, like that in SZ, use previously processed data to predict not yet processed data. That is, for a given prediction point $f_p(i,j)$ on a two-dimensional grid, this value is obtained using previous *upwind* values where the element indices are less than i and j. In our implementation, we are only recovering a single value rather than trying to compress a data stream; thus, we do not need to rely on previously processed points for our reconstruction. This allows us to use values from any direction in order to recover the corrupted value. In 1D, we have a left and right version of the prediction. In general, we have 2^d versions of the Lorenzo predictor for a d dimensional data array. Our Lorenzo recovery method utilizes preceding spatial neighbors to predict the corrupted value, unless preceding values are not available due to the index of the corrupted value.

3.4.6 Linear Regression. SZ-2.0 introduces a new predictor, linear regression. This predictor computes a series of coefficients based on the data being processed. The coefficients parameterize a regression prediction model that predicts the values to be compressed. We refer the reader to [20] for the details of the equations. When computing the regression coefficients, we exclude the corrupted index to avoid significantly impacting the prediction. Moreover, unlike the implementation in SZ, this reconstruction method uses the full dataset during reconstruction.

3.4.7 Local Linear Regression. The standard linear regression technique uses all the dataset. To lower the cost of this method, we create a localized version similar to that of SZ. The localized version overlays a patch of 3 layers in all dimensions around the corrupted datum ($V(i \pm 3, j \pm 3)$). We use the data values inside this patch to construct the regression coefficients. As with the full dataset version, we exclude the corrupted datum when computing the regression coefficients.

3.4.8 Lagrange Polynomial Interpolation. Lagrange polynomial interpolation creates a k degree polynomial that interpolates a given $k+1$ data points. Lagrange polynomials are used in a variety of applications in numerical analysis and are used in Reed-Solomon error correction codes. The Lagrange polynomial interpolation function of degree k for values y_i to y_{i+k} is:

$$L(x) = \sum_{r=0}^{k} y_i \ell_r(x), \tag{3}$$

where the basis functions $\ell_j(x)$ are defined as:

$$\ell_r(x) = \prod_{0 \le m \le k, m \ne k} \frac{x - x_m}{x_r - x_m}. \tag{4}$$

In our implementation, we use $k = 3$ data points around the corrupt element in the slowest changing dimension to construct the polynomial. The surrounding data points include two preceding values and one succeeding value.

4 EXPERIMENTAL RESULTS

4.1 Experimental Hardware and Application Test Data

To quantify the effectiveness of the SDC and DUE recovery algorithms from Section 3.4, we select 111 real-world data sets are from 5 applications on SDRBench [1]. We show the details of each application in Table 2.

All experiments are run on Clemson University's Palmetto Cluster, where we use nodes containing two Intel Xeon E5-2665 2.40GHz processors and 64 GB of memory. We compile all of our code using gcc version 8.5.0.

4.2 Experimental Design

The fault model assumes that hardware faults lead to bit-flips in memory locations. Furthermore, it is assumed that the location of the corruption is identified as a DUE by the machine check

Name	Domain	Data Dimensions	Data Set Count
Nyx	Cosmology	512 × 512 × 512	6
CESM-ATM	Climate	1800 × 3600	79
Miranda	Hydrodynamics	256 × 384 × 384	7
HACC	Cosmology	280953867	6
ISABEL	Climate	100 × 500 × 500	13

Table 2: Overview of applications we extract data sets from.

architecture or via a software based detector (see Section 3.1). Thus, we know the exact address where the corruption exists. We relate the address to a memory allocation before recovering the data (see Section 3.3).

In each of the experiments below, a fault injection campaign on at least 6000 trials is conducted on each dataset from each application. Each fault injection trial identifies at random indices of the corrupted datum. For each randomly selected value in the data set, that value is corrupted with a single bit-flip, and we evaluate each of the reconstruction methods. In particular, the relative error in the reconstructed data are compared to the original data values for each recovery method.

4.3 Analyzing Method Accuracy

The reconstruction methods outlined in Section 3.4 differ in both complexity and their reconstruction accuracy. To assess how well these methods reconstruct the corrupted datum, we compute the relative error between the true value and the reconstructed values. Depending on the end user's demands, the corrupted datum must be reconstructed to a specified accuracy level. To account for this, we investigate the relative error in the reconstructed datum at 3 levels: 1%, 5%, and 10%.

4.3.1 Overall Method Accuracy. Figure 2 showcases the overall accuracy of the methods when considering all applications together. Specifically, it demonstrates the percentage of predictions that produce a relative error less than 1%. From this figure, we see a clear dichotomy in the performance of the methods. Zero, Random, Linear Regression and Local Linear Regression successfully recover the corrupted datum 17% of the time. The other methods see reconstruction accuracy between 67-84%, with Lorenzo 1-Layer as the best method with 84%.

Relaxing the accuracy requirement on the reconstruction to a relative error within 5% (Figure 3) or 10% (Figure 4), all methods see improvement[3]. Most substantially, Local Linear Regression sees a 55% increase, going from 1% to 5%. We do not see as large of an improvement in Linear Regression because its accuracy is hampered by long range correlations across the full data set. Lorenzo 1-Layer continues to be the best reconstruction method, increasing to 90% and 92% for 5% and 10% relative error, respectively.

When looking at the three reconstruction accuracy levels, we see that the user's choice greatly determines which methods are feasible. In the extreme, as with Local Linear Regression, we go from completely discounting a method to it becoming a contender for best. Thus, we have to leverage multiple best methods. In the lossy compression domain, SZ leverages this to select the best predictor based on the data's properties and accuracy requirements [20]. We incorporate the selection between multiple best via auto-tuning.

[3]Related work in the area of lossy data compression consider these as acceptable distortions in the data [19]

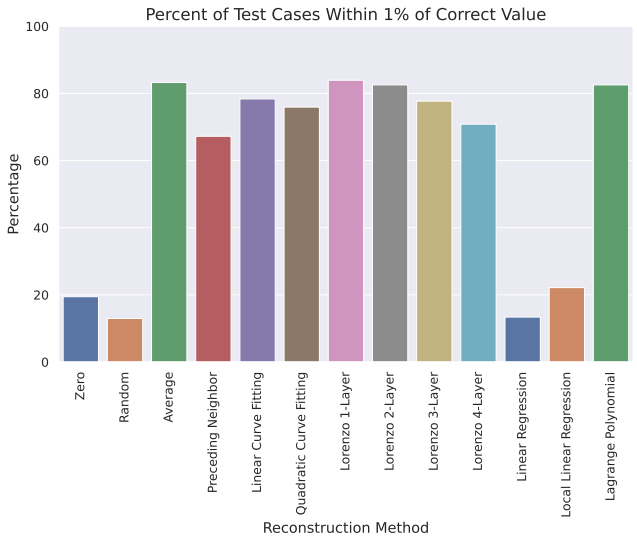

Figure 2: Method reconstruction with less than 1% relative error.

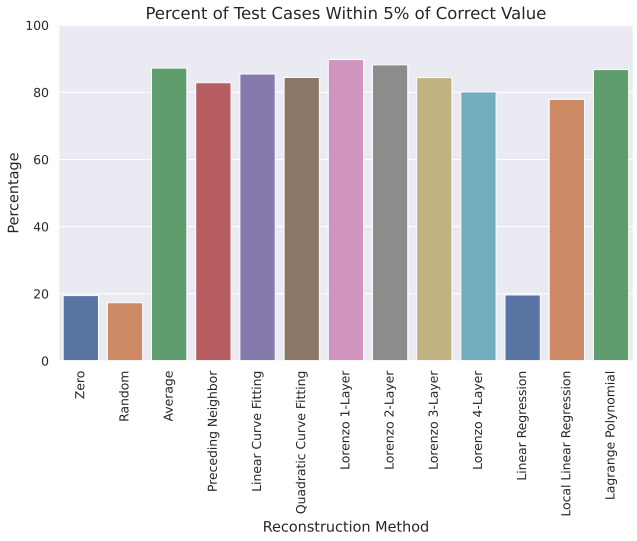

Figure 3: Method reconstruction with less than 5% relative error.

4.3.2 Application Specific Method Accuracy. Because of the variability in data values between applications, some applications are easier to recover data in than others. Figure 5 shows the breakdown of Figure 2 by application and highlights clear differences between them. The CESM application has the best prediction accuracy within the 1% relative error bound for most methods except Zero, Random, Linear Regression, and Local Linear Regression. These methods

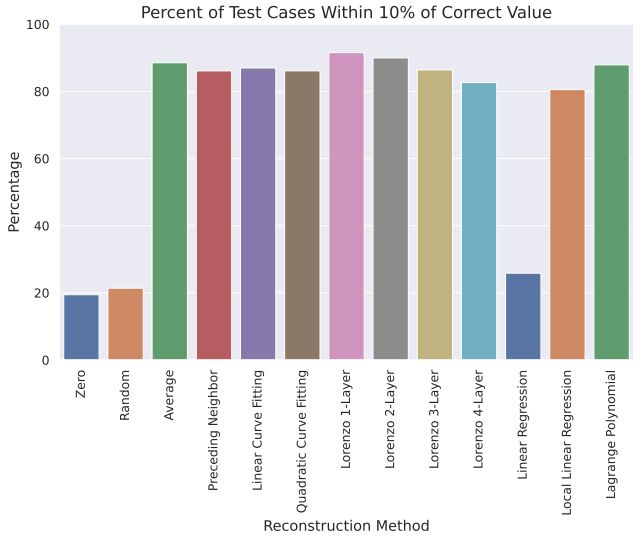

Figure 4: Method reconstruction with less than 10% relative error.

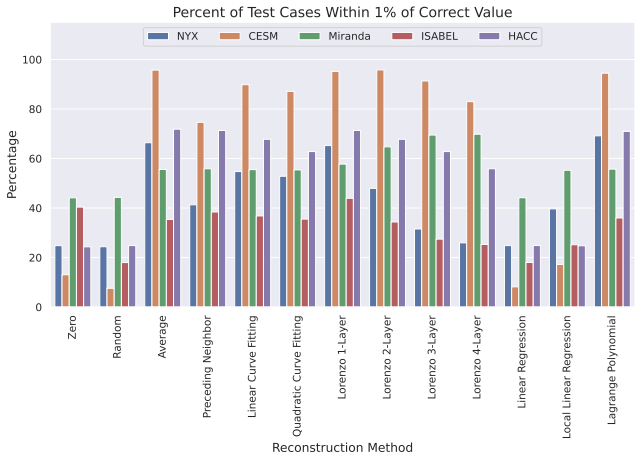

Figure 5: Method reconstruction with less than 1% relative error for each application.

correspond to the same methods that perform poorly in Figure 2. This is due to the disproportionate number of CESM datasets (see Table 2). For CESM, Average is the best method, indicating that spatially close values are very close in magnitude. Furthermore, Average performs well on all data sets except ISABEL, beating out much more computationally complex methods. When looking at the Zero method used by prior works [12, 13], we see poor performance on all data sets, indicating that zero, although an easy reconstruction algorithm, is not reliable.

Figure 6 and Figure 7 relax the relative error bound leads to increase the percent of trials that succeed. Here, trial success is defined as having a reconstructed value less than the error threshold. However, we still see similar application orderings for most

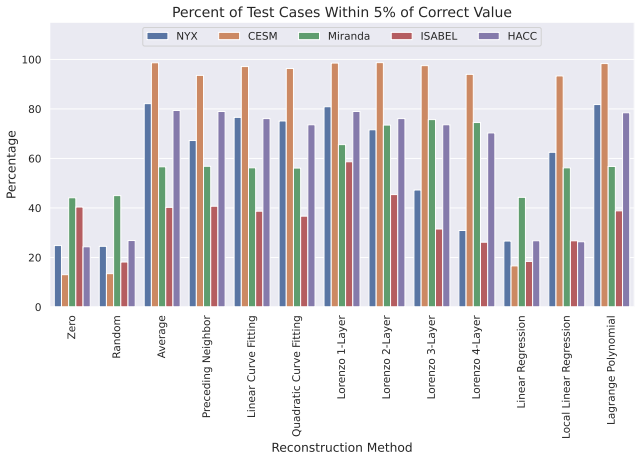

Figure 6: Method reconstruction with less than 5% relative error for each application.

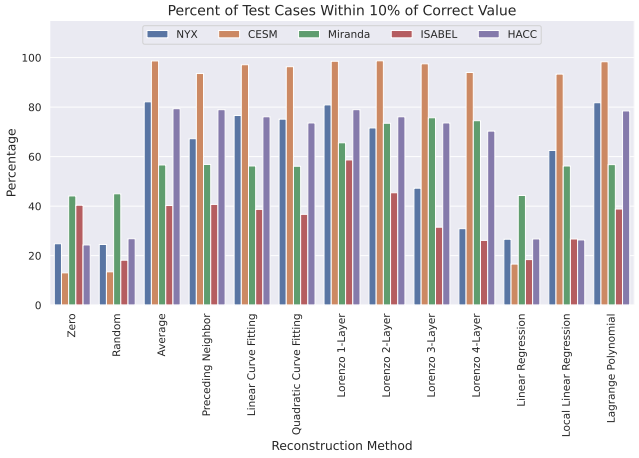

Figure 7: Method reconstruction with less than 10% relative error for each application.

methods. The biggest exception is Local Linear Regression, where CESM has the highest success percentage.

4.4 Auto-tuning

Because there is not a single best method for all applications, our auto-tuner searches for the best method in a spatially close region around the corrupted datum. To understand the effectiveness of the auto-tuner, we classify a tuning as successful if the reported method yields a relative error within the threshold[4]. Figure 8 shows the percent of trials where the auto-tuner's selection yields the best method, with $k = 3$ and with a 1% relative error. We select a 1% relative error bound because that is the toughest configuration and provides the best opportunity for the auto-tuner. The percentage for each application is comparable to the best methods in Figure 5.

[4]It does not have to yield the lowest relative error to be classified as a success. It only needs to be within the relative error tolerance.

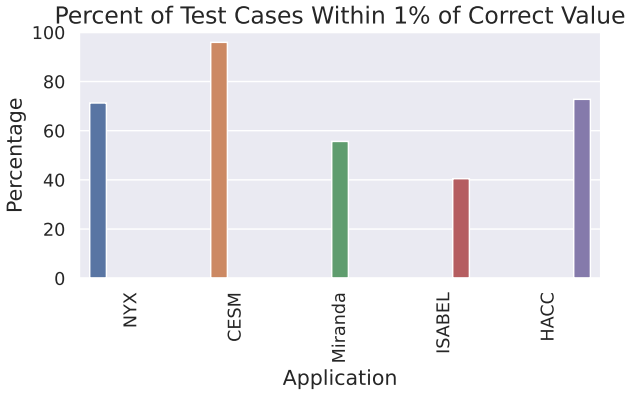

Figure 8: Accuracy of the local tuning algorithm with $k = 3$.

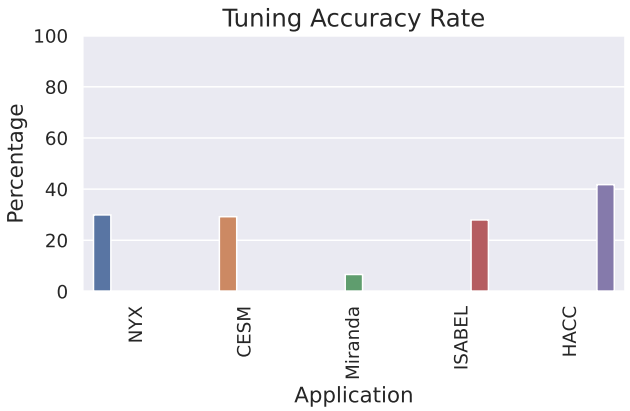

Figure 9: Percentage of trials where the auto-tuner selects the method that yields the lowest absolute relative error with $k = 3$.

The largest difference is 2% for NYX. For HACC and CESM, our tuning method yields a 1% and 0.1% improvement, respectively.

Although an auto-tuner may report a method that reconstructs the datum within the relative error threshold, it might have the lowest overall error. Figure 9 presents the probability that the auto-tuner selects the method that yields the smallest relative error. The auto-tuner gives the lowest relative error in HACC 42% of the time. However, on Miranda, the auto-tuner gives the lowest relative error 7% of the time.

4.5 Runtime Overhead

Although, accuracy in reconstruction is the primary success metric, we need to investigate the runtime cost associated with the methods and auto-tuning to ensure viability. All the reconstruction methods except for Linear Regression use a constant amount of data from the data sets regardless of the dimensions of the data set. Linear Regression requires all the elements of the data array. Given this, we select a single representative data set, *CLOUDf48* from ISABEL, to use in our experiments that quantify the runtime overhead of each method. To ensure accurate timings, we run each method in a loop a minimum of 10 times and ensure that the loop's total runtime is greater than 1 second.

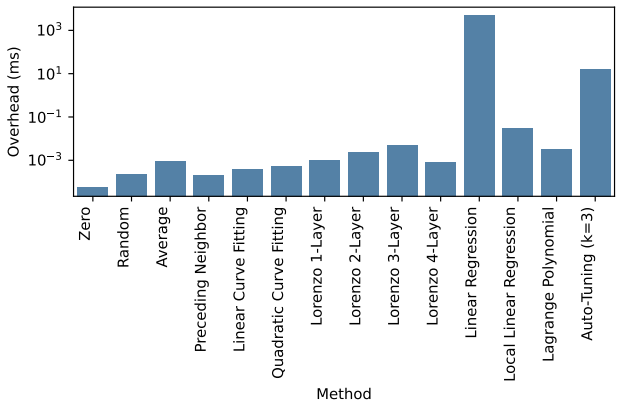

Figure 10: Runtime overhead for the reconstruction methods.

Figure 10 shows the runtime overhead for each method. Linear Regression has the largest runtime due to it accessing all the data elements. The remaining methods vary in overheads from 5e-5 ms (Zero) to 0.028 ms (Local Linear Regression). When considering auto-tuning, the overhead is 15.83 ms. However, the total overhead for auto-tuning is the sum of the auto-tuning time and the method execution time of what it classifies as the best. Results show that this is less than 15.86 ms. For these reconstruction methods, the overhead is small relative to HPC application runtime. Moreover, if we compare this method to checkpoint-restart, the overhead of spatial recovery is still small. Checkpoint-restart overhead is the time to recompute the work lost after the last checkpoint. On average, the restart overhead is half of the checkpointing interval, which can range from a few minutes to a few hours.

5 RELATED WORK

Many methods which leverage spatial and temporal smoothness have been developed to allow an HPC application in execution to detect and recover from faults. The authors in [24] use the invariant of global mass to detect whether a transient upset has occurred. In [3, 9], data values outside a prescribed range are identified as SDC via online monitoring. All of the preceding methods employ the use of checkpoint/restart to recover from an invalid state. In [26], corrupted data resulting from DUEs is recovered through a combination of error-correcting code output and application-specific context. In [13], a more general approach uses spatial averaging in memory to reconstruct an erroneous datum. In [12], DUEs are simply ignored, and the application is allowed to continue executing. Salloum et al. use linear interpolation to reconstruct outlier data elements in PDE computation [27]. This work proposes a novel forward-recovery method which leverages spatial data in all dimensions to allow for an accurate reconstruction of a value which has been corrupted, either by a DUE or via SDC.

6 CONCLUSION

Detectable uncorrectable errors and silent data corruption are high-risk corruption issues that can skew simulation results. While checkpoint-restart or unique application techniques are functional solutions, low-cost spatial recovery is a valuable combatant for DUE and SDC correction. Our approach utilizes multiple reconstruction methods. Each reconstruction method utilizes local data to reconstruct the corrupted value through prediction. Results show that the Lorenzo 1-Layer prediction method is the most accurate prediction method, with over half of its predictions within 1% of the correct value. However, discrepancies between individual reconstruction method accuracy decrease in proportion to the data set's spatial smoothness. (Data sets with greater spatial smoothness produce higher uniform accuracy.) Our auto-tuning allows for selecting the best method most of the time across applications. Moreover, the low runtime overhead compared to checkpoint-restart demonstrate that spatial recovery is effective in mitigating the negative influences of DUEs and SDC.

ACKNOWLEDGMENTS

Clemson University is acknowledged for generous allotment of compute time on the Palmetto cluster. This material is based upon work supported by the National Science Foundation under Grant No. SHF-1910197, SHF-1943114, and OAC-2204011.

REFERENCES

[1] [n. d.]. SDRBench. https://sdrbench.github.io/

[2] Emmanuel Agullo, Luc Giraud, Abdou Guermouche, Jean Roman, and Mawussi Zounon. 2016. Numerical recovery strategies for parallel resilient Krylov linear solvers. *Numerical Linear Algebra with Applications* 23, 5 (2016), 888–905.

[3] Leonardo Bautista-Gomez and Franck Cappello. 2015. Detecting Silent Data Corruption for Extreme-Scale MPI Applications. In *Proceedings of the 22Nd European MPI Users' Group Meeting* (Bordeaux, France) *(EuroMPI '15)*. ACM, New York, NY, USA, Article 12, 10 pages. https://doi.org/10.1145/2802658.2802665

[4] L. Bautista-Gomez and F. Cappello. 2015. Exploiting Spatial Smoothness in HPC Applications to Detect Silent Data Corruption. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. 128–133. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.9

[5] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka. 2011. FTI: High performance Fault Tolerance Interface for hybrid systems. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12. https://doi.org/10.1145/2063384.2063427

[6] Jon Calhoun, Luke Olson, Marc Snir, and William D. Gropp. 2015. Towards a More Fault Resilient Multigrid Solver. In *Proceedings of the Symposium on High Performance Computing* (Alexandria, Virginia) *(HPC '15)*. Society for Computer Simulation International, San Diego, CA, USA, 1–8. http://dl.acm.org/citation.cfm?id=2872599.2872600

[7] Jon Calhoun, Marc Snir, Luke N. Olson, and William D. Gropp. 2017. Towards a More Complete Understanding of SDC Propagation. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing* (Washington, DC, USA) *(HPDC '17)*. ACM, New York, NY, USA, 131–142. https://doi.org/10.1145/3078597.3078617

[8] Marc Casas, Bronis R. de Supinski, Greg Bronevetsky, and Martin Schulz. 2012. Fault Resilience of the Algebraic Multi-Grid Solver. In *Proceedings of the 26th ACM International Conference on Supercomputing* (San Servolo Island, Venice, Italy) *(ICS '12)*. Association for Computing Machinery, New York, NY, USA, 91–100. https://doi.org/10.1145/2304576.2304590

[9] Sheng Di and Franck Cappello. 2016. Adaptive Impact-Driven Detection of Silent Data Corruption for HPC Applications. *IEEE Trans. Parallel Distrib. Syst.* 27, 10 (Oct. 2016), 2809–2823. https://doi.org/10.1109/TPDS.2016.2517639

[10] Sheng Di and Franck Cappello. 2016. Fast Error-Bounded Lossy HPC Data Compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, IL, USA, May 23-27, 2016*. 730–739. https://doi.org/10.1109/IPDPS.2016.11

[11] James Elliott, Mark Hoemmen, and Frank Mueller. 2014. Evaluating the Impact of SDC on the GMRES Iterative Solver. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS '14)*. IEEE Computer Society, Washington, DC, USA, 1193–1202. https://doi.org/10.1109/IPDPS.2014.123

[12] Bo Fang, Qiang Guan, Nathan Debardeleben, Karthik Pattabiraman, and Matei Ripeanu. 2017. LetGo: A Lightweight Continuous Framework for HPC Applications Under Failures. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing* (Washington, DC, USA) *(HPDC '17)*. ACM, New York, NY, USA, 117–130. https://doi.org/10.1145/3078597.3078609

[13] Bo Fang, Hassan Halawa, Karthik Pattabiraman, Matei Ripeanu, and Sriram Krishnamoorthy. 2019. BonVoision: Leveraging Spatial Data Smoothness for Recovery from Memory Soft Errors. In *Proceedings of the ACM International Conference on Supercomputing* (Phoenix, Arizona) *(ICS '19)*. Association for Computing Machinery, New York, NY, USA, 484–496. https://doi.org/10.1145/3330345.3330388

[14] Lorenzo Ibarria. 2007. *Geometric Prediction for Compression*. Ph. D. Dissertation. USA. Advisor(s) Rossignac, Jarek. AAI3271523.

[15] Intel. 2021. *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2*. Technical Report. Intel.

[16] Luc Jaulmes, Marc Casas, Miquel Moretó, Eduard Ayguadé, Jesús Labarta, and Mateo Valero. 2015. Exploiting Asynchrony from Exact Forward Recovery for DUE in Iterative Solvers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Austin, Texas) *(SC '15)*. ACM, New York, NY, USA, Article 53, 12 pages. https://doi.org/10.1145/2807591.2807599

[17] Himanshu Kaul, Mark Anders, Steven Hsu, Amit Agarwal, Ram Krishnamurthy, and Shekhar Borkar. 2012. Near-Threshold Voltage (NTV) Design: Opportunities and Challenges. In *Proceedings of the 49th Annual Design Automation Conference* (San Francisco, California) *(DAC '12)*. Association for Computing Machinery, New York, NY, USA, 1153–1158. https://doi.org/10.1145/2228360.2228572

[18] Kuang-Hua Huang and J. A. Abraham. 1984. Algorithm-Based Fault Tolerance for Matrix Operations. *IEEE Trans. Comput.* C-33, 6 (1984), 518–528. https://doi.org/10.1109/TC.1984.1676475

[19] Xin Liang, Sheng Di, Dingwen Tao, Zizhong Chen, and Franck Cappello. 2018. An Efficient Transformation Scheme for Lossy Data Compression with Point-Wise Relative Error Bound. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. 179–189. https://doi.org/10.1109/CLUSTER.2018.00036

[20] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. 2018. Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets. In *2018 IEEE International Conference on Big Data (Big Data)* (Seattle, WA, USA, 2018-12). IEEE, 438–447. https://doi.org/10.1109/BigData.2018.8622520

[21] R. E. Lyons and W. Vanderkulk. 1962. The Use of Triple-Modular Redundancy to Improve Computer Reliability. *IBM Journal of Research and Development* 6, 2 (1962), 200–209. https://doi.org/10.1147/rd.62.0200

[22] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R. de Supinski. 2010. Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–11. https://doi.org/10.1109/SC.2010.18

[23] Xiang Ni, Esteban Meneses, Nikhil Jain, and Laxmikant V. Kale. 2013. ACR: Automatic Checkpoint/Restart for Soft and Hard Error Protection. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '13)*. IEEE Computer Society.

[24] D. Nicholaeff, N. Davis, D. Trujillo, and R. W. Robey. 2012. *Cell-Based Adaptive Mesh Refinement Implemented with General Purpose Graphics Processing Units*. Technical Report. Los Alamos National Laboratory – Eulerian Codes.

[25] B. Nicolae, A. Moody, E. Gonsiorowski, K. Mohror, and F. Cappello. 2019. VeloC: Towards High Performance Adaptive Asynchronous Checkpointing at Large Scale. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 911–920.

[26] Alexandra Poulos, Dylan Wallace, Robert Robey, Laura Monroe, Vanessa Job, Sean Blanchard, William Jones, and Nathan DeBardeleben. 2018. Improving Application Resilience by Extending Error Correction with Contextual Information. In *Proceedings of the31st International Conference on High Performance Computing, Networking, Storage and Analysis (SC) Workshops 2018:8th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS) 2018*. IEEE Computer Society, Los Alamitos, CA, USA, Dallas, TX, USA, 19–28. https://doi.org/10.1109/FTXS.2018.00006

[27] Maher Salloum, Jackson R. Mayo, and Robert C. Armstrong. 2016. In-Situ Mitigation of Silent Data Corruption in PDE Solvers. In *Proceedings of the ACM Workshop on Fault-Tolerance for HPC at Extreme Scale* (Kyoto, Japan) *(FTXS '16)*. Association for Computing Machinery, New York, NY, USA, 43–48. https://doi.org/10.1145/2909428.2909433

[28] Marc Snir, Robert W Wisniewski, Jacob A Abraham, Sarita V Adve, Saurabh Bagchi, Pavan Balaji, Jim Belak, Pradip Bose, Franck Cappello, Bill Carlson, et al. 2014. Addressing failures in exascale computing. *The International Journal of High Performance Computing Applications* 28, 2 (2014), 129–173.

[29] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. 2017. Significantly Improving Lossy Compression for Scientific Data Sets Based on Multidimensional Prediction and Error-Controlled Quantization. In *2017 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2017, Orlando, FL, USA, May 29 - June 2, 2017*. 1129–1139. https://doi.org/10.1109/IPDPS.2017.115

[30] John W. Young. 1974. A First Order Approximation to the Optimum Checkpoint Interval. *Commun. ACM* 17, 9 (Sept. 1974), 530–531. https://doi.org/10.1145/361147.361115