Lightweight Silent Data Corruption Detection Based on Runtime Data Analysis for HPC Applications

Eduardo Berrocal
Illinois Institute of Technology
Chicago, IL
USA
eberroca@iit.edu

Leonardo Bautista-Gomez
Argonne National Laboratory
Argonne, IL
USA
leobago@anl.gov

Sheng Di
Argonne National Laboratory
Argonne, IL
USA
sdi1@anl.gov

Zhiling Lan
Illinois Institute of Technology
Chicago, IL
USA
lan@iit.edu

Franck Cappello
Argonne National Laboratory
Argonne, IL
USA
cappello@anl.gov

ABSTRACT

Next-generation supercomputers are expected to have more components and, at the same time, consume several times less energy per operation. Consequently, the number of soft errors is expected to increase dramatically in the coming years. In this respect, techniques that leverage certain properties of iterative HPC applications (such as the *smoothness* of the evolution of a particular dataset) can be used to detect silent errors at the application level. In this paper, we present a pointwise detection model with two phases: one involving the prediction of the next expected value in the time series for each data point, and another determining a range (i.e., normal value interval) surrounding the predicted next-step value. We show that dataset correlation can be used to detect corruptions indirectly and limit the size of the data set to monitor, taking advantage of the underlying physics of the simulation. Our results show that, using our techniques, we can detect a large number of corruptions (i.e., above 90% in some cases) with 84% memory overhead, and 13.75% extra computation time.

Index Terms—Fault Tolerance, Resilience, High-Performance Computing, Silent Data Corruption, Soft Errors, Time Series

1. INTRODUCTION

High-performance computing (HPC) is changing the way scientists make discoveries. Science applications require ever-larger machines to solve problems with higher accuracy. While future systems promise to provide the power needed to tackle those science problems, they are also raising new challenges. For example, transistor size and energy consumption of future systems must be significantly reduced. Such steps might dramatically impact the soft error rate (SER) according to recent studies [4].

HPDC'15, June 15-19, 2015, Portland, OR, USA
©2015 ACM 978-1-4503-3550-8/15/06 $15.00
DOI: http://dx.doi.org/10.1145/2749246.2749253.

Random memory access (RAM) devices have been intensively protected against silent data corruption (SDC) through error-correcting codes (ECCs) because they have the largest share of the susceptible surface on high-end computers. Recent studies, however, indicate that ECCs alone cannot correct an important number of DRAM errors [10]. In addition, not all parts of the system are ECC-protected: in particular, logic units and registers inside the processing units are usually not protected by ECC but by other methods because of the space, time, and energy cost that ECC requires in order to work at low level. Historically, the SER of central processing units was minimized through a technique called *radiation hardening*, which consists of increasing the capacitance of circuit nodes in order to increase the critical charge needed to change the logic level. Unfortunately, this technique involves increasing either the size or the energy consumption of the components, which is prohibitively expensive at extreme scale.

Runtime data analysis can be used to leverage the fact that some datasets produced by iterative HPC applications (i.e., the applications' state at a particular point in time) evolve *smoothly* from one time step to the next. This characteristic can be used effectively to design a general SDC detection scheme with relatively low overhead. In particular, we will show that an interval of *normal* values for the evolution of the datasets can be predicted, such that any corruption will *push* the corrupted data point outside the expected interval of *normal* values, and it will, therefore, become an *outlier.*

Previous work in this area have tried to solve the SDC problem using different strategies. Namely: by hardware-level detection [11], process replication [7], algorithm-based fault tolerance (ABFT) [9], and approximate computing [3]. These techniques, however, are either too expensive resource-wise, such as hardware-level detection (energy intensive) or process replication (2x or 3x in extra resources), or not general enough, such as ABFT or approximate computing (algorithms need to be adapted manually and only a subset of the kernels can be protected).

In our previous work [5, 2, 6], we showed the feasibility of using data analysis with simple and lightweight linear predictors to detect SDC efficiently. In [6] we used *error-*

feedback control to improve prediction recall (See Equation (2) for a definition of recall), and a technique called *even sampling* to reduce memory overhead.

In this paper, we introduce a new detection model based on the user-expected accuracy and past prediction errors. In addition, we perform a comprehensive evaluation during live executions with two popular DoE applications: Nek5000 and HACC. We also observe that it is viable to detect, using the underlying physics of the simulation, corruptions in one dataset by just using another, thus opening the door for memory saving strategies.

The rest of the paper is organized as follows. In Section 2 we present our proposed detector. In Section 3 we present our evaluation and results. Finally, in Section 4 we summarize our key findings and future work.

2. ANOMALY DETECTION

Our pointwise SDC detection model has two phases: the first phase involves the prediction of the next expected value in the time series for each data point, while the second determines a range (i.e., normal value interval) surrounding the predicted next-step value. Soft errors can be detected by observing whether a particular value falls outside this computed range. The range size, or normal value interval, will play an important role in obtaining high precision and recall, defined in Equations (1) and (2), respectively. Here *TP*, *FP*, and *FN* refer to *true positives*, *false positives*, and *false negatives*, respectively.

$$precision = \frac{TP}{TP + FP} \quad (1)$$

$$recall = \frac{TP}{TP + FN} \quad (2)$$

The magnitude of the range size depends on the relative location of the predicted value and the user-expected accuracy. The key notations used to formulate the pointwise detection

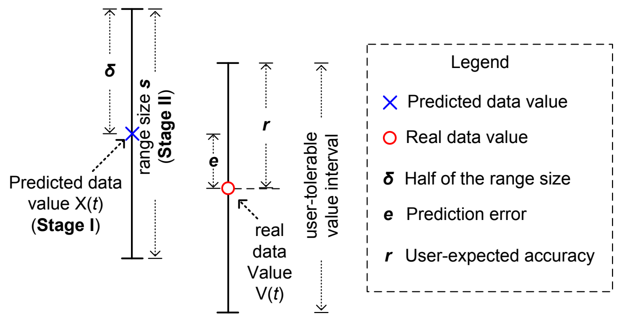


Figure 1: Illustration of our one-step prediction model (at time step t).

model are presented in Figure 1. $X(t)$ is the predicted value at time step t. The prediction error (denoted by e) is equal to the difference between the predicted value $X(t)$ and the real data value (denoted by $V(t)$) computed at the current time step. In practice, we find that $\delta = r + e$ works well (zero false positives) when predictions are very good (i.e., $e < r$). Here r represents the user-expected accuracy. The real error e, however, is unknown at runtime, so we approximate it by using the last-step prediction error (see Equation (3)), since we have observed that prediction errors at adjacent time steps exhibit a high degree of autocorrelation.

$$\hat{e}(t) = |V(t-1) - X(t-1)| \quad (3)$$

The first predictor, called linear curve fitting (LCF), uses the two most recent previous time steps to fit a linear curve, which is then projected to the next time step in order to predict the next value in the time series. Equation (4) shows how this prediction is calculated. $\Delta(t-1)$ is the slope of the curve (velocity) at time $t-1$.

$$\begin{aligned} X(t) &= \Delta(t-1) + V(t-1) \\ &= (V(t-1) - V(t-2)) + V(t-1) \\ &= 2V(t-1) - V(t-2) \end{aligned} \quad (4)$$

The acceleration-based predictor (ABP)[1] uses the two and three most recent previous time steps to extract the velocity ($\Delta(t-1)$) and acceleration ($\Delta^2(t-1)$) of the data, respectively, and then combines them to compute the prediction for the next value in the time series.

$$\begin{aligned} X(t) &= \Delta^2(t-1) + \Delta(t-1) + V(t-1) \\ &= 3V(t-1) - 3V(t-2) + V(t-3). \end{aligned} \quad (5)$$

The autoregressive (AR) and the autoregressive moving average ($ARMA$) models assume that every value in the time series depends linearly on its previous values. Equation (6) describes AR, where c is a constant, φ_i are the coefficients of the model, p is the number of coefficients, and $\varepsilon(t) \sim \mathcal{N}(0, \sigma^2)$ is the noise at time t. Similarly, Equation (7) describes ARMA, which adds the moving average part to AR. The errors $\varepsilon(t-i)$ are computed by using the past prediction errors:

$$\varepsilon(t-i) = V(t-i) - X(t-i)$$

The coefficients φ_i and θ_i (we set $p = q = 4$) are computed by using the first 10 time steps of the simulation by least squares with the Yule-Walker equations. We assume that no errors occur during this period; otherwise the coefficients would not reflect the reality of the application's data behavior.

$$X(t) = c + \sum_{i=1}^{p} \varphi_i V(t-i) + \varepsilon(t) \quad (6)$$

$$X(t) = c + \sum_{i=1}^{p} \varphi_i V(t-i) + \varepsilon(t) + \sum_{i=1}^{q} \theta_i \varepsilon(t-i) \quad (7)$$

3. EVALUATION

In this section, we present a set of experimental results to verify the efficacy of our SDC detector in production-level iterative HPC applications. All our predictors, as well as our bit-flip fault injector, are already implemented transparently in the fault tolerance interface (FTI) toolkit [1] to protect the execution against silent data corruptions.

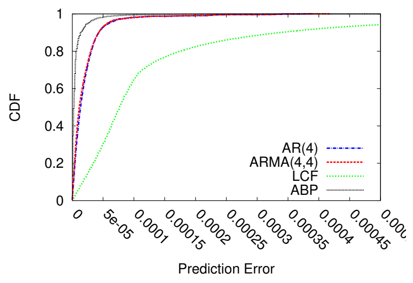
We evaluate two well-known and widely used DoE HPC applications: Nek5000 [12] (a CFD kernel), and HACC [8] (an N-body cosmology application). Nek5000 is a CFD solver based on the spectral element method. It is also being used for a large number of applications in diverse fields such as reactor thermal-hydraulics and biofluids. HACC (for Hybrid/Hardware Accelerated Cosmology Code) is a cosmology code aimed at understanding the nature of dark matter and dark energy in the universe.

3.1 Prediction Errors

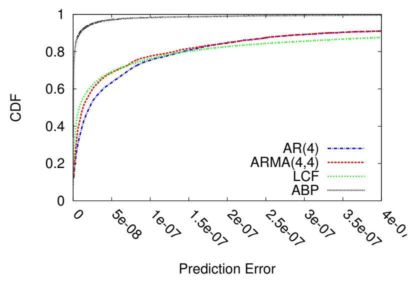
We characterize the distribution of the prediction errors under different predictors with the two mentioned iterative

[1]It can also be called quadratic curve fitting (QCF).

HPC application traces regarding different variables. The variables inlcude the position's coordinates (x) of the particles in HACC, and the vertical flow and pressure (in a large eddy simulation) for Nek5000. We performed the prediction at each time step for each data point, measuring the error by taking the absolute difference between the real value and the predicted one: $e(t) = |V(t) - X(t)|$. Each trace involves millions of prediction results, which allow us to build a cumulative distribution function (CDF) of the prediction error e, as shown in Figure 2.



(a) HACC (particles' position)



(b) Nek5000 (vortex)

Figure 2: Cumulative distribution function of prediction errors for different predictors and HPC datasets.

The most interesting result from these experiments is that a relatively simple predictor such as *ABP* is able to achieve smaller prediction errors than more complex linear models such as the well-known *AR* and *ARMA* models for the selected HPC applications. In absolute terms, for the HACC application, up to 90% of the predictions have an error less than or equal to 10^{-5} under ABP, whereas only 8% to 56% of other predictors can reach such low errors. In the case of Nek5000, the prediction errors (90% of predictions) for vorticity and pressure are less than or equal to 8×10^{-9} and 2.8×10^{-10}, respectively. By comparison, other predictors are not able to achieve errors below 10^{-7}. In addition, the *AR* and *ARMA* models require more memory sizes per data point (since they need to store the coefficients), and a coefficient learning phase for a number of time steps (in our case 10) in the training period.

3.2 Detection Results

In the second set of experiments, we test our detector using traces and real application runs. We choose the ABP predictor because it has the smallest prediction error, as seen in Section 3.1. In the case of real application runs, we run HACC with 512 MPI ranks and around 16 million particles, protecting the position and velocity variables. Nek5000 is run with 64 MPI ranks and a grid of 573,440 data points per rank. Seven variables are protected: position(x,y,z), velocity(vx,vy,vz), and pressure.

In Figure 3 we inject bit-flips at random particles and/or data points on particular bit positions on different datasets. In the case of HACC (single-precision datasets), we set $r = 10^{-6}$; and for Nek5000 (double-precision datasets), we set $r = 10^{-8}$. In the figure, *vel* refers to injection and detection on the particles' velocity dataset, and *pos+vel* refers to injection on velocity while *detecting on position.* In the latter case (Figure 3(b)), we want to explore the idea of leveraging datasets' correlation for detection (i.e., having a corruption in one dataset visible by the other). In the case of HACC, *velocity* is used to move a particle to a new *position.*

Three conclusions can be extracted from these results. First, if we consider only the corruptions outside the range of the user-expected accuracy, our method can cover over 90% of all possible corruptions for HACC (Figure 3(a)). Similarly, our method can cover 66% of corruptions for Nek5000 (Figure 3(c)).

Second, the performance of our detector depends heavily on the underlying dataset (Figure 3(b)). We have observed that position changes are smoother than velocity changes, thus making the next values for velocity more difficult to predict. The good news is that leveraging datasets correlations works. Apart from the savings in memory overhead, these results indicate that we can achieve a similar recall by monitoring only position, rather than monitoring both.

Third, we see that our predictors have different results depending on whether we work with traces or real application runs. The reason for such disparities is that traces do not represent the totality of the application's data state and are used only to construct distributions to help us understand different predictors and parameters. In any case, the results are indeed similar enough to make us confident in our experiments using traces.

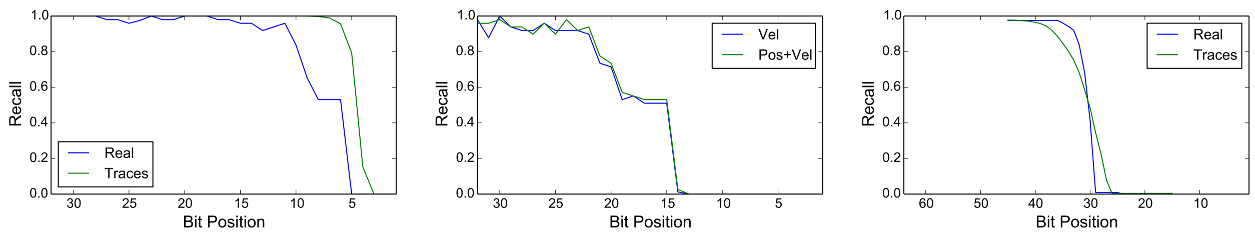
3.3 Performance Overhead

Our SDC detector (using ABP as predictor) is always below 90% memory overhead, 15% runtime overhead, and has a 0% extra network communication for both applications. By comparison, 2x-replication will produce an overhead of 100% in all three dimensions. The overheads for HACC are 84% extra memory consumption and 13.75% extra computation time. For Nek5000, the memory overhead is only 9.5%, while the extra computation time never goes above 1%.

4. CONCLUSION AND FUTURE WORK

We have developed a new model to tackle the problem of SDC detection using user-expected accuracy and past prediction errors. We compared a large number of linear predictors that take advantage of the characteristics of iterative HPC application datasets, and also implemented and evaluated our detector model with production-level scientific applications using both traces and real experiments on supercomputers. The detection results are promising. Considering only the corruptions outside the range of the user-expected accuracy, our method can cover 90% of all possible corruptions for HACC (single-precision), and over 66% for Nek5000 (double-precision).

We have shown that it is viable to detect corruptions in one variable (such as velocity) by using another (such as position) in one of our applications (HACC), taking advantage of the fact that these variables are interconnected by the underlying physics of the application. Based on this initial observation,



(a) HACC (particles' position) (b) HACC (velocity) (c) Nek5000 (vortex)

Figure 3: Comparing recall for bit-flips injected at random particles and/or data points.

we plan to explore further the use of variable correlation to reduce the memory requirement of detection while keeping the detection recall high for all variables.

One limitation of our approach is the number of false positive (FP) detections. Although the FP rate is always below 10^{-4} for HACC, and always below 10^{-3} for Nek5000, this is still too high for a production level SDC detection tool. Consider that the large number of detections involved (one per data point, with millions of potential points per application) would produce a *positive* in every iteration. There are two ways in which this problem can be solved: (1) making our predictors more accurate, and (2) by optimizing the detection range parameter δ to find the optimum value that would maximaze *recall* by producing almost zero FP. We plan to follow these two paths in our future research.

Another limitation is performance overheads (see Section 3.3), especially for applications such as HACC that are memory bound. For such cases, we plan to investigate compression methods to group close points together in order to reduce the memory footprint significantly without incurring in a big reduction in detection recall.

We note that these techniques would only work for iterative HPC applications. Although such applications are the majority today, especially in the scientific area of computational physics, other types of applications are becoming important in the HPC community in areas such as computational biology and statistics. Also, simulations with abrupt variable changes, like applications involving collisions, may need different types of detectors. For those, much work remains to be done.

Acknowledgments

This material was based upon work supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program, under Contract DE-AC02-06CH11357, and also in part by the ANR RESCUE, and INRIA-Illinois Joint Laboratory for Petascale Computing. The work at Illinois Institute of Technology was supported in part by US National Science Foundation grant CNS-1320125 and CCF-1422009. The software used in this work was developed in part by the DOE NNSA ASC- and DOE Office of Science ASCR-supported Flash Center for Computational Science at the University of Chicago

5. REFERENCES

[1] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka. Fti: High performance fault tolerance interface for hybrid systems. In *SC'11*, pages 32:1–32:32, 2011.

[2] L. A. Bautista-Gomez and F. Cappello. Detecting silent data corruption through data dynamic monitoring for scientific applications. In *PPoPP'14*, pages 381–382, 2014.

[3] A. R. Benson, S. Schmit, and R. Schreiber. Silent error detection in numerical time-stepping schemes. *International Journal of High Performance Computing Applications*, pages 1–20, 2014.

[4] S. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25:10–16, Nov. 2005.

[5] S. Di, E. Berrocal, L. Bautista-Gomez, K. Heisey, R. Guptal, and F. Cappello. Toward effective detection of silent data corruptions for hpc applications. SC '14 - poster, 2014.

[6] S. Di, E. Berrocal, and F. Cappello. An efficient silent data corruption detection method with error-feedback control and even sampling for hpc applications. CCGRID, 2015.

[7] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell. Detection and correction of silent data corruption for large-scale high-performance computing. In *SC'12*, pages 78:1–78:12, 2012.

[8] S. Habib, V. A. Morozov, H. Finkel, A. Pope, K. Heitmann, K. Kumaran, T. Peterka, J. A. Insley, D. Daniel, P. K. Fasel, N. Frontiere, and Z. Lukic. The universe at extreme scale: Multi-petaflop sky simulation on the bg/q. In *SC'12*, pages 1–11, 2012.

[9] K.-H. Huang and J. A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers*, 100(6):518–528, 1984.

[10] A. A. Hwang, I. A. Stefanovici, and B. Schroeder. Cosmic rays don't strike twice: Understanding the nature of dram errors and the implications for system design. In *ASPLOS'XVII*, pages 111–122, 2012.

[11] S. S. Mukherjee, J. Emer, and S. K. Reinhardt. The soft error problem: An architectural perspective. In *HPCA'05*, pages 243–247. IEEE, 2005.

[12] J. Shin, M. W. Hall, J. Chame, C. Chen, P. F. Fischer, and P. D. Hovland. Speeding up nek5000 with autotuning and specialization. In *ICS'10*, pages 253–262, 2010.